CONTINGENCY PLANNING AND OBSTACLE ANTICIPATION FOR AUTONOMOUS DRIVING

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Jason Scott Hardy

August 2013

CONTINGENCY PLANNING AND OBSTACLE ANTICIPATION FOR AUTONOMOUS DRIVING

Jason Scott Hardy, Ph.D.

Cornell University 2013

This thesis explores the challenge of robustly handling dynamic obstacle uncertainty in autonomous driving systems. The path planning performance of Cornell's autonomous vehicle platform Skynet in the DARPA Urban Challenge (DUC) is analyzed and a new contingency planning formulation is presented that incorporates anticipated obstacle motions for improved collision avoidance capabilities. A discrete set of trajectory predictions is generated for each dynamic obstacle in the environment based on possible maneuvers the obstacle might make. A set of contingency paths is then optimized in real-time to accurately account for the mutually exclusive nature of these obstacle predictions. Computational scaling is addressed using a trajectory clustering algorithm that allows the contingency planner to plan a fixed number of paths regardless of the number of dynamic obstacles and possible obstacle goals in the environment. This contingency planning approach is evaluated using a series of human-in-the-loop experiments and simulations and is found to offer significant improvements in safety compared to the DUC planner and in performance compared to non-contingency planning approaches.

A method for performing multi-step prediction over a two-stage Gaussian Process (GP) model is also presented. This prediction method is applied to a two-stage driver-vehicle obstacle model for the generation of high quality obstacle motion predictions using observed obstacle trajectories. An on-the-fly data

selection technique is used to minimize computation when analytically evaluating higher order moments of the GP output. An adaptive Gaussian mixture model approach is also presented that allows this prediction technique to accurately predict the motion of highly nonlinear and multimodal systems.

BIOGRAPHICAL SKETCH

Jason Hardy earned his Bachelor of Science degree in Mechanical Engineering from the University of California Santa Barbara in June 2007. He earned a Masters of Science in Aerospace Engineering from Cornell University in February 2011. He is completing his Ph.D. in Aerospace Engineering at Cornell University under Professor Mark Campbell.

This thesis is dedicated to Mary and Oliver.

ACKNOWLEDGEMENTS

Thank you to Dr. Mark Campbell for guiding my graduate studies and for encouraging me to pursue a research direction that I find fascinating.

Thank you to my committee members, Dr. Hadass Kress-Gazit and Dr. Daniel Huttenlocher, as well as Dr. Noah Snavely for providing excellent advice and for encouraging active research on Cornell's autonomous vehicle platform.

Thank you to Dr. Jonathan Schoenberg, Frank Havlak, and Theo Park for all of your help getting my Skynet tests off the ground. Also, thank you Frank for all of your collaboration on obstacle anticipation modeling.

Thank you to the anonymous members of the Autonomous Systems Lab for always being willing to volunteer for my experiments.

Thank you to ARO Grant #W911NF-09-1-0466 and the Air Force Office of Scientific Research and Department of Defense for funding my work

Thank you to Skynet for allowing me to put you in dangerous collision avoidance scenarios that no self-respecting intelligent vehicle would ever find itself in.

Thank you to my parents for continuously supporting my education.

Finally, thank you to my wife Mary for putting up with six Ithaca winters with me and for encouraging and supporting me every step of the way.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Autonomous driving systems promise to revolutionize automobile safety by mitigating user error and by anticipating and proactively responding to impending collisions. Autonomous driving systems must be able to identify obstacles, plan routes, and safely interact with dynamic agents. Current solutions are adept at performing these tasks in structured environments with well defined rules, as demonstrated by the 2007 DARPA Urban Challenge (DUC) [10]. Since the DUC, autonomous systems have advance to the point of extensive live testing on public roads. Google's driverless car project [29] is testing autonomous driving on public roads in both California and Nevada. Autonomous Convoying experiments have traversed large stretches of public roads accross Europe and Asia [9],[71]. This growing interaction between autonomous driving systems and human driven vehicles highlights the need for safe, reliable motion planning and collision avoidance. Reactionary planning systems and brittle deterministic obstacle handling logic break down when an autonomous vehicle encounters complex, unforeseen obstacle behaviors [20]. This thesis is presented as a series of papers that address the questions of how to codify defensive driving and how to balance safety, performance, and scalability in highly uncertain dynamic environments.

Chapter 2 presents an overview of the trajectory planner used by Cornell's autonomous vehicle platform Skynet in the DARPA Urban Challenge (DUC). This work is derived from Hardy et al. [33]. The contributions from Chapter 2 are:

- Provides an overview of the trajectory planning system used by Cornell's autonomous vehicle platform Skynet during the DUC.
- Presents a sensitivity study analyzing the robot's planning performance as a function heuristic weights in the trajectory planner. This study is conducted using a data driven simulation based on logged data from the semi-finals of the DUC.
- Presents analysis of key obstacle avoidance events during the semi-finals and final competition of the DUC.

Chapter 3 presents an optimization based trajectory planner capable of planning multiple contingency paths to directly account for uncertainties in the future trajectories of dynamic obstacles. This work is derived from Hardy and Campbell [32]. The contributions from Chapter 3 are:

- The simultaneous optimization of partially shared contingency trajectories over a robot's continuous planning space in order to accurately handle mutually exclusive obstacle predictions.
- A computationally efficient and accurate upper bound on point-wise collision probability between two polygonal objects with uncertain relative position and orientation.
- A spline based trajectory representation and associated cost function and constraint definitions for expressing the contingency path planning problem using only a small number of optimization variables.
- An obstacle trajectory clustering algorithm designed to reduce planning complexity by capturing the most important mutually exclusive obstacle decisions using a limited number of obstacle trajectory clusters.

- Simulation results and performance analysis for single obstacle, as well as congested and less-congested multiple obstacle interactions.
- A sensitivity analysis of the collision probability threshold constraint.

Chapter 4 presents the results for a series of human-in-the-loop experiments and simulations to evaluate the performance of obstacle anticipation and contingency planning during live two-way interactions with human driven obstacle vehicles. This work is derived from Hardy et al. [35]. The contributions from Chapter 4 are:

- A virtual obstacle interface is developed in order to provide a safe framework for testing planning decisions during dangerous collision avoidance scenarios.
- Experimental results and analysis are presented for a series of live experiments using a human-in-the-loop virtual obstacle vehicle.
- Simulation results and analysis using prerecorded human driven obstacle data is used to study the effects of timing on the relative planning performance of the tested planning approaches.
- Computational scaling analysis is presented for simulation results with three simultaneous obstacles using prerecorded human driven obstacle data.
- Simulation results and analysis are presented for a series of human-in-the-loop simulations for both single obstacle and multiple obstacle scenarios.

Chapter 5 presents techniques for performing multi-step prediction of a two-stage Gaussian Process (GP) regression model. These techniques are applied to

a two-stage driver-vehicle model trained using data collected in a driving simulation environment. This work is derived from Hardy et al. [34]. The contributions from Chapter 5 are:

- A method for performing multi-step prediction using a two stage model is presented in which GP regression is used to model complex or unknown system dynamics such as an intelligent controller and a parametric model is used to model well understood system dynamics. Analytical moment evaluations are used to compute the mean and variance of the GP output, as well as the cross covariance between the GP output and the initial system state distribution used as an input.
- An on-the-fly data selection technique is presented to enable computationally efficient analytical evaluations of higher order moments of a GP output using only the most relevant subset of training data for a given input distribution.
- An adaptive Gaussian mixture model (GMM) formulation is presented to allow the analytical GP propegation techniques to better model highly nonlinear and multimodal system behaviors.
- A two-stage driver-vehicle model is presented using a GP regression model for driver commands and a parametric vehicle model for known vehicle dynamics.
- Prediction results are presented for the driver-vehicle model using driver data collected from human volunteers navigating a four-way intersection in a driving simulation environment. These results compare the analytical propagation method with alternative extended Kalman filter (EKF) and sampling based methods.

- Computational Scaling results are presented as a function of the on-the-fly reduced data set size.
- Prediction results for multimodal compound driving behaviors are presented using the adaptive GMM prediction approach.

CHAPTER 2

OPTIMIZATION-BASED TRAJECTORY PLANNER FOR AUTONOMOUS VEHICLES IN URBAN ENVIRONMENTS

Abstract

This paper presents the derivation, sensitivity study, and experimental implementation of a nonlinear optimization based path planner for large scale robots operating in complex, dynamic, urban environments. The path planner utilizes a novel mixture of discrete and continuous path planning steps to facilitate a safe, smooth, and predictable driving behavior. The planner first locates a drivable corridor through a robot-centered obstacle map using a discrete grid based graph search algorithm. An optimal path is then planned through this corridor using a cost based nonlinear optimization routine with both hard and soft constraints. The behavior of this optimization is influenced by tunable weights which govern the relative cost contributions assigned to different path characteristics. The path planner was experimentally implemented in Cornell University's 2007 DARPA Urban Challenge robot Skynet. The sensitivity of the robot's performance to the path planner weights is studied using a simulation based on logged data from the semi-finals of the 2007 DARPA Urban Challenge. The performance of the path planner in selected areas of both the semi-finals and the final Urban Challenge is also presented and analyzed.

2.1 Introduction

Navigation in a complex urban environment requires the ability to sense, track, and identify obstacles, make decisions, and plan and execute paths both at the abstract road network level and at the local trajectory planning level. While robotic planning has received much attention over the years, the urban environment problem is more challenging for several reasons. First, the environment is dynamic, with moving obstacles, which renders many classical planning methods unworkable. Second, while there are many constraints and rules (such as driving laws), the ability to plan as uncertainties arise (e.g. during a road blockage) is critical. Finally, as vehicle speeds increase, these problems, as well as computational demands, require more innovative solutions to work in real time.

Numerous classical path planning techniques have been developed using discrete grid searches [77] and potential field methods [50], which can reliably find traversable paths through a locally stored obstacle or terrain map. In a discrete grid search, the robot's obstacle/terrain map is overlaid with a predetermined grid and a discrete search algorithm such as A* or Dijkstra's algorithm is performed to identify an optimal path along the grid [77]. Generally, these methods are unable to cope with the complexities of a full sized mobile robotic systems in complex environments. They can have difficulty generating smooth paths, and in handling nonholonomic robot dynamics and large robot dimensions. In potential field methods [46, 83], a continuous potential function is formulated which rewards progress towards the goal state and penalizes closeness to obstacles. The robot can then follow the gradient of the potential function to the goal state [50]. These path planning methods are effective for fully actu-

ated point robots in simple environments, however they are typically difficult to implement for complex robot dynamics and can suffer from saddle points [50].

To address path planning for robots with nonholonomic dynamics operating in a complex environment, maneuver libraries were developed where a predetermined set of maneuvers is evaluated, and the best maneuver, as evaluated based on safety and goal completion criterion, is executed. This approach is especially effective in structured environments such as multi-lane roads and variations of it were used by Stanford in the 2005 DARPA Grand Challenge [76] and the 2007 DARPA Urban Challenge (DUC) [64], as well as by Tartan Racing in the 2007 DUC [78]. Despite its success, the maneuver library approach breaks down in unstructured scenarios such as zone travel (where no road lanes are defined) or blockage recovery (where road boundaries can be ignored) because there is no guarantee that an acceptable path lies within the set of predetermined maneuvers. Thus, these approaches typically require a large amount of hand tuning, and, many times, an alternate planner is necessary to handle unstructured scenarios.

To allow for robots with complex dynamics to track these planned paths, the pure pursuit [12] and vector pursuit [86] algorithms were developed. A pure pursuit controller computes the instantaneous curvature required to move the robot to a point a specified lookahead distance down the path [12]. Similarly, a vector pursuit controller computes the instantaneous screw motion (rotation about and translation along a vector centerline) required to move the robot to a given lookahead point while also matching the point's orientation [86].

Current successful path planners for full sized robotic systems have evolved along two main approaches: randomized sampling and optimization based ap-

proaches. Randomized approaches, such as the rapidly exploring random tree (RRT) [53, 52] use randomized sampling to quickly explore the search space. RRTs have been shown to work in real time on complex robotic systems, and were used by MIT in the 2007 DUC [48]. The drawback to these randomized sampling approaches is that it can be very difficult to establish an efficient sampling strategy for a wide variety of environments. Proposal distributions must be tuned rigorously to avoid both undersampling drivable regions and wasting samples on undrivable or oversampled regions. Additionally, since the search space is randomly sampled, there are no guarantees of performance or even convergence within the allotted planning time.

Unlike randomized approaches, optimization based path planners rely on discrete or continuous optimization strategies to explore the search space. Discrete search techniques are typically applied on the robot's input space in order to account for nonlinear and nonholonomic robot dynamics. Stanford's Hybrid A* planner [64] and Tartan Racing's Anytime D* planner [78] both use unconstrained discrete search techniques and were both successfully implemented as auxiliary zone planners in the 2007 DUC. However, these discrete planners can be computationally expensive, making them not well suited to perform as a sole, or primary path planner. Continuous optimization based planners use numerical nonlinear optimization techniques to compute an optimal path subject to vehicle constraints over a short planning horizon. This optimization strategy was implemented by Caltech for a static, desert terrain in the 2005 DARPA Grand Challenge [13].

This paper details a path planner for large scale, nonholonomic robots operating in dynamic, urban environments, based on a nonlinear optimization

method. The presented path planner was implemented on Cornell University's 'Skynet' robot (Figure 2.1), one of six autonomous robots to complete the 2007 DUC, executing 56 miles of autonomous driving in a populated urban environment in under six hours. The formulation of this planning problem for the nonlinear solver is unique for several reasons. First, orthogonal search vectors and velocities are used to define the path search space, which makes the solver very fast and easy to implement. Second, physical constraints are easily added to the problem, such as maximum path curvature and obstacle avoidance. Third, the cost is formulated as a combination of several penalty functions, such as penalties on acceleration/deceleration and curvature; these were chosen to yield predictable, human-like driving behaviors. This approach is similar to planning approaches that have been used to model human driving behavior. For example, in [67], human driving is modeled by optimizing a projected path using a combination of inequality constraints and penalty functions, including similar penalties on forward and lateral accelerations, speed, and distance from lane boundaries. While this problem formulation requires a nonlinear solver, the simplicity of the search vectors enables fast, predictable outputs in real time. Two additional contributions of this paper are the design details and analysis of the successful experimental implementation of the planner in the 2007 DUC, and a sensitivity study of the planner to the cost function weights in order to yield intuition about how the optimization parameters influence the path planning performance.

The outline of this paper is as follows. Section 2.2 describes the general architecture and data flow in Skynet's low-level planning layer. Section 2.3 summarizes the theoretical elements of the path planner, and the setup of the problem for the nonlinear optimization routine. Section 2.4 details the performance of

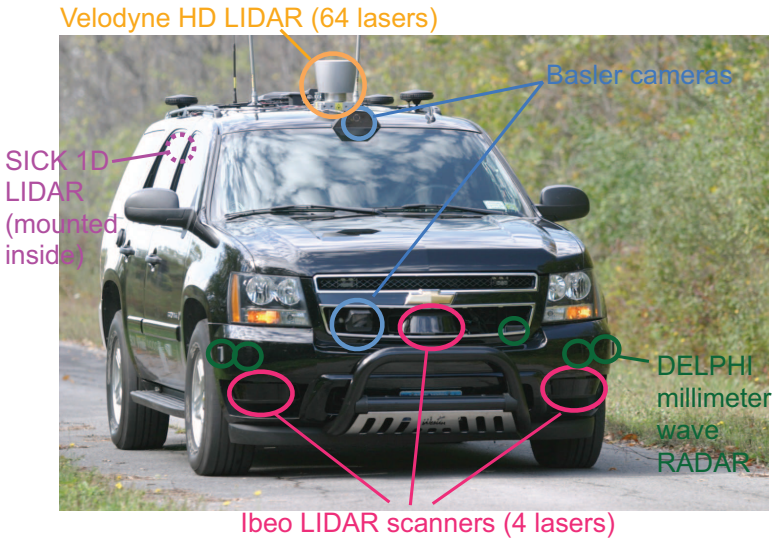


Figure 2.1: Cornell University's DARPA Urban Challenge entry: Skynet.

the path planner in Skynet during the DUC, including an analysis of several scenarios, such as a vehicle traveling the wrong way on a one way road, an obstacle course of closely spaced parked cars, and a reverse maneuver. Section 2.5 analyzes the sensitivity of the robot's driving style to its designated driving preferences, formulated as a set of optimization weighs.

2.2 Operational Layer Architecture and Data Flow

Skynet's hierarchical planner was designed to execute smooth, intelligent, predictable behaviors in the presence of uncertainties. Trajectory planning through Skynet's local environment is performed at the lowest level of this hierarchy, called the Operational Layer. The Operational Layer utilizes localization information and vehicle-relative obstacle information in order to generate actuator commands which obey driving behaviors dictated by more abstract layers of the planner (Figure 2.2). The Operational Layer processes each obstacle estimate (a

point cloud of laser returns) from Skynet's probabilistic obstacle tracker [56] into a convex hull. An obstacle map of these convex hulls is then searched to identify a drivable corridor, and required spacings from obstacles and lane/road boundaries are cast as constraints with respect to the vehicle. A nonlinear trajectory optimization is then used to plan an optimal path through this corridor, under constraints, in real time.

The architecture and data flow of the Operational Layer is shown in Figure 2.2. This planning sequence consists of a preprocessing phase, a trajectory optimization phase, and a tracking and actuation phase, which are briefly described here to provide context before the formulation of the trajectory optimizer.

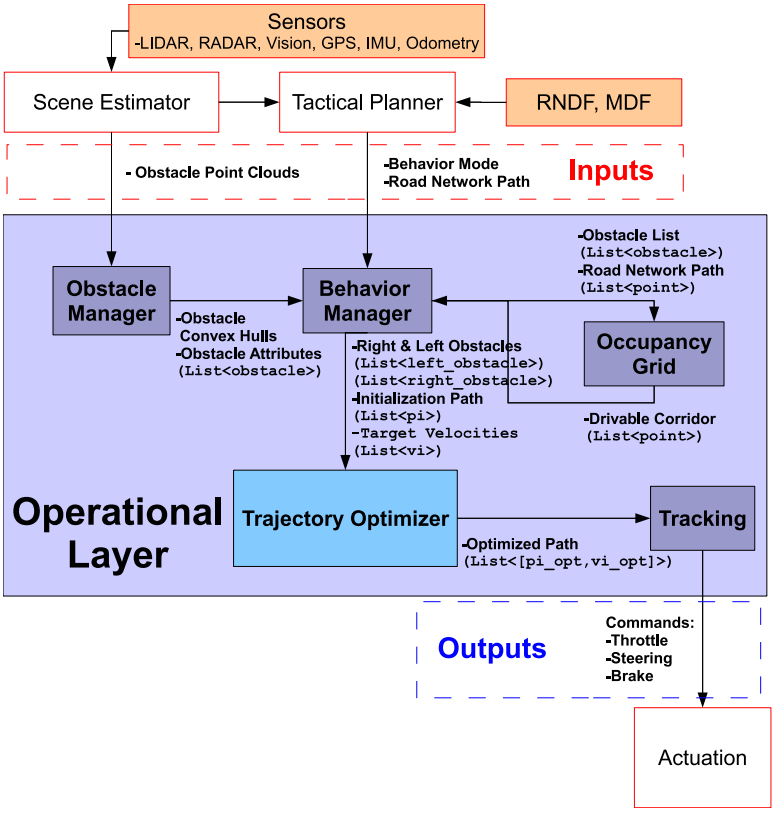


Figure 2.2: Diagram of the Operational Layer's Architecture and Data Flow.

2.2.1 Operational Layer Inputs

The Operational Layer takes as inputs obstacle information from the Scene Estimator and high level planning information from the Tactical Planner. The Scene Estimator is a formal Bayesian estimator consisting of two components, a localization component and an obstacle tracking component, which supply the Operational Layer with probabilistic estimates of Skynet's position, velocity, attitude, and nearby obstacles [60]. The localization component consists of a bootstrap particle filter that fuses position information from multiple sensors over time into a robust and accurate estimate of Skynet's position and heading on the road network [57]. This component utilizes multiple sources of available positioning information: absolute positioning information provided by a tightly-coupled GPS and inertial navigation system (INS), and in-lane positioning information provided by lane and stop line detection algorithms implemented on Skynet's two forward-facing optical cameras [58, 60]. The final localization signal is passed to Skynet's Tactical Planner and Operational Layer at 10 Hz.

The second component of the Scene Estimator, the obstacle tracking component, consists of a Rao-Blackwellized particle filter that jointly estimates the number of nearby obstacles and their states: position, speed, heading, and shape [56]. This estimator fuses information over time from Skynet's obstacle sensors: seven laser rangefinders, eight millimeter-wave radars, and two optical cameras. The resulting obstacle estimates are also augmented with metadata estimated for each obstacle: a unique and persistent identification number, whether the obstacle is stopped, whether the obstacle is car-like based on size, and whether all or part of the obstacle is occluded by other objects in the environment. Information for each of these tracked obstacles is passed to the Tactical

Planner and Operational Layer at 10 Hz, along with a list of untracked obstacle points belonging to objects that are too small or too large to be moving.

The Operational Layer's other main input source, the Tactical Planner, is responsible for processing mission goal information from the road map, defined in a Route Network Definition File (RNDF) and mission objectives, defined in a Mission Definition File (MDF). The Tactical Planner combines this a priori information with environment and position information from the Scene Estimator to decide on an appropriate high level behavior mode, and to plan a route through the RNDF specified road network [60]. These behavior modes include basic driving modes such as "stay in lane" behavior for normal road travel, or "change lane" or "U-turn" behaviors for specific vehicle maneuvers. The Operational Layer is then responsible for finding a local path through nearby obstacles while advancing the robot down the RNDF route specified by the Tactical Planner.

2.2.2 Operational Layer Data Flow: Preprocessing

The Operational Layer's preprocessing phase consists of three modules, the Obstacle manager, the Behavior Manager, and the Occupancy Grid, as shown in Figure 2.2. Obstacle information is received from the Scene Estimator in the form of laser data point clouds representing sensed obstacles. The obstacle information is received by the Obstacle Manager in a vehicle-relative coordinate system to eliminate any coupling between uncertainty in obstacle positions and uncertainty in the robot's global position. In the Obstacle Manager, this obstacle information is converted from a point cloud representation to a polygon repre-

sentation using a convex hull algorithm [27]. These new obstacle polygons are assigned a minimum and desired spacing based on their Scene Estimator specified metadata; for example a small static obstacle is assigned a minimum spacing of 0.2 meters and a desired spacing of 0.4 meters, while a dynamic car-like obstacle is assigned a minimum spacing of 0.75 meters and a desired spacing of one meter.

The obstacle polygons generated by the obstacle manager are sent to the Behavior Manager, which also receives the higher-level planning information sent by the Tactical Planner. The Behavior Manager is ultimately responsible for orchestrating the Operational Layer's execution based on the behavior mode specified by the Tactical Planner. A zone travel behavior, for example, indicates that no lane boundary information can be extracted from the RNDF map and the planning goals should be to advance to the route planned destination while avoiding sensed obstacles. If problems arise such as a complete road blockage, the Behavior Manager communicates with the Tactical Planner to find and implement a solution. These solutions typically consist of either a change in behavior mode or a route replan. A detailed description of the Tactical Planner and its different behavior modes is given in Miller et al. [60].

As a preparatory step for the Trajectory Optimizer, the Behavior Manager sends all of its obstacle polygon information, along with a goal state, to an Occupancy Grid module. The Occupancy Grid module constructs a discrete binary obstacle occupancy grid relative to the vehicle and computes a path to the goal state using a modified version of A* [66]. The rough path produced by this algorithm, is never driven directly; rather it is used to define a drivable corridor in which the trajectory optimization will be confined. This corridor path is sent

back to the Behavior Manager, where it is used to group obstacles to the left or right side of the path.

2.2.3 Operational Layer Data Flow: Trajectory Optimization, Tracking and Actuation

The Behavior manager sends an initialization path, represented as an (x,y) point list, an initial velocity profile, represented as a list of point velocities, and lists of right side and left side obstacle polygons, to the Trajectory Optimizer. The position portion of this initialization path is constructed by interpolating between the RNDF or mission waypoints and adjusting the points to fit within the specified drivable corridor. In the Trajectory Optimizer, this initialization path is refined into a final, optimal, drivable path. A depiction of how this initialization path relates to the RNDF and to the final optimized path is shown in Figure 2.3. Details of the Trajectory Optimizer's operation are covered in detail in Section 2.3.

The optimized path produced by the Trajectory Optimizer is then sent to the Tracking module where it is used to calculate actuation commands such as steering angle, throttle, and brake pressure. The Tracking module's operation is covered in more detail in Section 2.3. The actuation commands produced by the Tracking module are then sent to the Actuation Layer where they are executed on the vehicle.

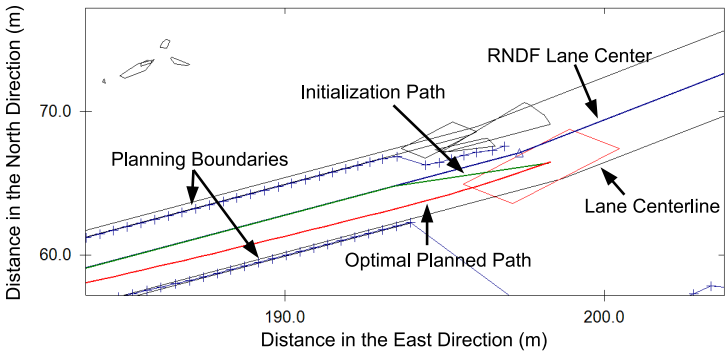


Figure 2.3: A snapshot of the Trajectory Optimizer's operation from the NQE depicting the RNDF, the shifted initialization path, and the final optimized path

2.3 Trajectory Optimizer

The Operational Layer's smooth, predictable paths are generated by the Trajectory Optimizer, which uses a nonlinear trajectory optimization algorithm to produce a physically drivable path subject to actuator constraints and obstacle avoidance. Figure 2.4 depicts the important variables used in the Trajectory Optimizer. The algorithm first resamples the initial path into a set of n equally-spaced base points $p_i = (x_i, y_i)$, $i \in [1, n]$. This path originates from the robot's current location. A set of n unit-length 'search vectors' u_i, $i \in [1, n]$ perpendicular to the initial path are also created, one for each base point. The Trajectory Optimizer then attempts to find a set of optimized path points $z_i = p_i + w_i \cdot u_i$, $i \in [1, n]$ by adjusting search weights w_i, $i \in [1, n]$. Target velocity magnitudes v_i, $i \in [1, n]$ are also considered for each point, as well as a set of variables q_i^{L} and q_i^{R}, $i \in [1, n]$ indicating the distance by which each path point z_i violates desired spacings on the left and right of the robot created by the list of polygonal obstacles sent from the Behavior Manager.

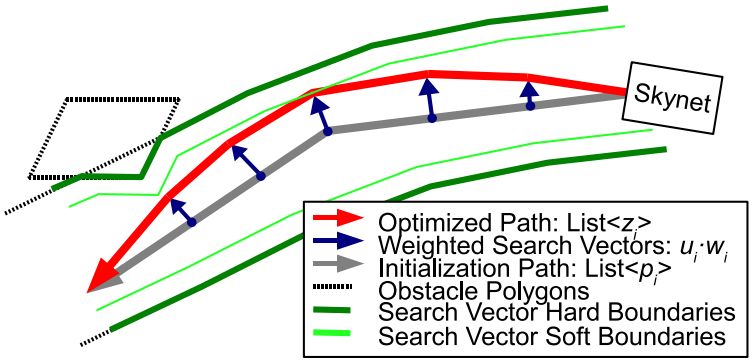


Figure 2.4: Diagram of the Trajectory Optimizer, with associated parameter definitions.

Search weights, velocities, and final obstacle spacings for all points $(1:n)$ are chosen to minimize the cost function J, formulated as a weighted sum of penalties derived from undesirable driving behaviors:

$$\begin{aligned} J\left(w_{1:n}, v_{1:n}, q^{\mathrm{L}}_{1:n}, q^{\mathrm{R}}_{1:n}\right) = \\ \alpha_c \sum_{i=2}^{n-1} \|c_i\|^2 + \alpha_d \sum_{i=2}^{n-2} \left(\|c_{i+1}\| - \|c_i\|\right)^2 + \alpha_w \sum_{i=1}^{n} \left(w_i - w_i^{\mathrm{t}}\right)^2 \\ + \alpha_q \sum_{i=1}^{n} \left(q_i^{\mathrm{L}} + q_i^{\mathrm{R}}\right) + \alpha_a \sum_{i=1}^{n-1} a_i^2 - \alpha_v \sum_{i=1}^{n} v_i \end{aligned} \tag{2.1}$$

where α_c, α_d, α_w, α_q, α_a, and α_v are tuning weights that specify the relative importance of driving behaviors, c_i is the approximated path curvature, w_i^{t} is the path target weight which penalizes deviations from specified target path points, and a_i is the approximated forward vehicle acceleration - all at the i^{th} path point.

The optimized cost function in Equation 2.1 has six terms, each corresponding to a particular undesirable driving behavior. The parameter α_c penalizes large curvatures, which correspond to undesirably sharp turns; α_d penalizes rapid changes in curvature, which correspond to unstable swerving; α_w penalizes large deviations from the target path offset w_i^{t}, which forces the robot to

move closer to the boundary of its allowed driving region; α_q penalizes violations of desired obstacle spacing; α_a penalizes sharp accelerations and braking; and α_v penalizes slow velocities, encouraging faster plan completion time. Adjustments made to the relative weighting of these penalty terms determine the robot's driving behavior, such as whether it prefers aggressive maneuvers or smoother driving.

The cost function in Equation 2.1 requires the calculation of path curvature and acceleration. The curvature k_i at each discretized path point is given by:

$$k_i = \frac{2\left(z_{i-1} - z_i\right) \times \left(z_{i+1} - z_i\right)}{\left\|z_{i-1} - z_i\right\| \cdot \left\|z_{i+1} - z_i\right\| \cdot \left\|z_{i+1} - z_{i-1}\right\|} \tag{2.2}$$

To simplify differentiation, the following approximate curvature term is used instead:

$$c_i = \left(z_{i-1} - z_i\right) \times \left(z_{i+1} - z_i\right) \tag{2.3}$$

This approximation is made noting that 1) the initial path points p_i are equally spaced, and 2) the search weights w_i are constrained to be small, so the denominator of Equation 2.2 is approximately equal for all path points and can be absorbed into the weight α_c. The approximate forward vehicle acceleration a_i is also calculated under a similar approximation:

$$a_i = \frac{v_{i+1}^2 - v_i^2}{2 \cdot \left\|p_{i+1} - p_i\right\|} \tag{2.4}$$

The cost function $J\left(w_{1:n}, v_{1:n}, q_{1:n}^{\mathrm{L}}, q_{1:n}^{\mathrm{R}}\right)$ presented in Equation 2.1 is optimized subject to a set of six rigid path constraints of the form $C_1\left(w_{1:n}, v_{1:n}, q_{1:n}^{\mathrm{L}}, q_{1:n}^{\mathrm{R}}\right) = 0$ or $C_2\left(w_{1:n}, v_{1:n}, q_{1:n}^{\mathrm{L}}, q_{1:n}^{\mathrm{R}}\right) < 0$:

1. The path must begin at the robot's current location and heading:

$$w_1 = 0 \tag{2.5}$$

$$w_2 = w_2^{\theta_{init}}; \tag{2.6}$$

where $w_2^{\theta_{init}}$ is constrained to lie on the initial heading vector defined by θ_{init}, the robot's current orientation.

2. Each search weight w_i cannot push the smoothed path outside the boundary polygon supplied by the Tactical Layer:

$$\begin{aligned} [w_i]_{\min} - w_i &< 0 \\ w_i - [w_i]_{\max} &< 0 \quad \forall i \in [1, n] \end{aligned} \tag{2.7}$$

3. Each obstacle spacing variable q_i^{L} and q_i^{R} cannot exceed any obstacle's minimum spacing requirement:

$$\begin{aligned} q_i^{\mathrm{L}} - \left[q_i^{\mathrm{L}}\right]_{\max} &< 0 \\ q_i^{\mathrm{R}} - \left[q_i^{\mathrm{R}}\right]_{\max} &< 0 \quad \forall i \in [1, n] \end{aligned} \tag{2.8}$$

4. Each curvature k_i, calculated using Equation 2.2, cannot exceed a maximum turning curvature:

$$\|k_i\| - [k]_{\max} < 0 \quad \forall i \in [2, n-1] \tag{2.9}$$

5. Total forward and lateral vehicle acceleration at each path point cannot exceed maximum limits, as defined by the acceleration ellipse:

$$\left(\frac{a_i}{\left[a^{\mathrm{F}}\right]_{\max}}\right)^2 + \left(\frac{\|k_i\| \cdot v_i^2}{\left[a^{\mathrm{L}}\right]_{\max}}\right)^2 - 1 < 0 \tag{2.10}$$

where $\left[a^{\mathrm{F}}\right]_{\max}$ is the maximum allowed forward acceleration and $\left[a^{\mathrm{L}}\right]_{\max}$ is the maximum allowed lateral acceleration.

6. The difference between consecutive search weights w_i and w_{i+1} must not exceed a minimum and maximum:

$$\begin{aligned} [\Delta w]_{\min} - (w_{i+1} - w_i) &< 0 \\ (w_{i+1} - w_i) - [\Delta w]_{\max} &< 0 \quad \forall i \in [1, n-1] \end{aligned} \tag{2.11}$$

Additional constraints on the final path heading are also occasionally included to restrict the smoothed path to a particular end orientation. This is used, for example, when it is desired that the path end parallel to a lane or parking spot.

The constrained trajectory optimization problem is solved using LOQO [79], an off-the-shelf nonlinear convex optimization library. Two optimization passes are made to reach a final optimized path. The second pass uses the output of the first pass as an initial path which allows for a smoother final path and reduces the Trajectory Optimizer's dependency on the shape and discretization of the initialization path.

The final optimized path is then sent to the Tracking module where it is tracked by two independent low-level tracking controllers, one for desired speed and one for desired curvature. The speed controller is a proportional-integral (PI) controller with feedback linearization to account for engine and transmission inertia, rolling inertia, wind resistance, and power loss in the torque converter [24, 26, 87]. The curvature controller uses an Ackermann steering model with cornering stiffness to convert desired curvature into desired steering wheel angle, which is then passed as a reference signal to a proportional-integral-derivative (PID) steering wheel angle controller [26, 87]. This controller only feeds back on path curvature: lateral offset is not used as a feedback signal, because all paths are planned from the robot's current location and tracked in a vehicle-centric coordinate frame. The optimization is restarted from scratch at each planning cycle, and is run at 10 Hz on a dual core 2.0 GHz Pentium-Mobile processor running Windows Server 2003.

2.4 Performance in the DARPA Urban Challenge

The DUC semifinal round, known as the National Qualifying Event (NQE), was held in Victorville, California, USA from October 25, 2007 to October 31, 2007 at the Southern California Logistics Airport. At the NQE, 35 Urban Challenge robots were tested on three courses, called 'Area A,' 'Area B,' and 'Area C.' Each area tested one or more specific aspects of autonomous urban driving with focused and carefully-monitored scenarios. Only one robot was tested at a time, and any necessary traffic was provided by human-driven vehicles in preset patterns. Safe and sensible driving was paramount in the NQE, with stability and repeatability emphasized in the structure of the event. Teams that passed the NQE were invited to the Urban Challenge Event (UCE) which was held in Victorville, California, USA on November 3, 2007. Of the 35 teams originally invited to participate in the NQE, only 11 were invited to continue on to the UCE.

Throughout the course of the NQE and the UCE, there arose several scenarios which tested and exemplified the capabilities of Skynet's Operational Layer. Section 2.4.1 evaluates Skynet's performance on a portion of the NQE Area B course known as the 'Guantlet' where Skynet was forced to navigate a narrow road lined with parked cars and centerline barrels. Section 2.4.2 evaluates an incident during one of Skynet's early missions in the UCE, in which a confused human driver drove the wrong way up a one way road, forcing Skynet to avoid an unanticipated obstacle. Section 2.4.3 evaluates an incident that occurred during NQE Area A where Skynet found itself stuck against a concrete barrier on the side of the road and was able to safely plan and execute a reverse maneuver, backing back into its lane in order to allow it to continue its mission.

2.4.1 NQE Area B (The Gauntlet)

In the NQE Area B, the competing robots were required to navigate a lengthy route through the urban environment. The environment was devoid of moving traffic, but contained an area with parking spaces. One portion of the course, nicknamed the 'Gauntlet,' was also filled with orange barrels in the center of the street and cars parked at the side of each lane to test obstacle avoidance capabilities. The course also contained a number of empty intersections, and the robots were required to remain in the appropriate lane at all times.

Skynet ran through Area B twice, failing at the first attempt and succeeding at the second. Skynet failed its first attempt at Area B while trying to park. It successfully aligned itself to the parking space, but refused to pull into the space because other unoccupied cars parked nearby violated minimum obstacle spacing constraints, $[q_i]_{\max}$ in Equation 2.8. After sitting for several minutes, Skynet was manually stopped, positioned inside the parking space, and allowed to continue. Skynet proceeded as normal, entering the Gauntlet portion of the course. There, Skynet passed several parked cars and orange barrels, until coming upon a section of road with a parked car in either lane. Due to Skynet's conservative minimum obstacle spacings it believed that both lanes were blocked. Skynet began a U-turn maneuver, but ran out of time before completing the course. All behavior until the parking difficulty was normal, and Skynet navigated without incident.

Behavior in the first attempt at Area B prompted the team to make adjustments to spacing constraints in the optimization problem presented in Section 2.3. In particular, minimum spacing constraints, used to compute $[q_i]_{\max}$ in Equation 2.8, were reduced from 0.9 m to 0.3 m for stopped obstacles, and from

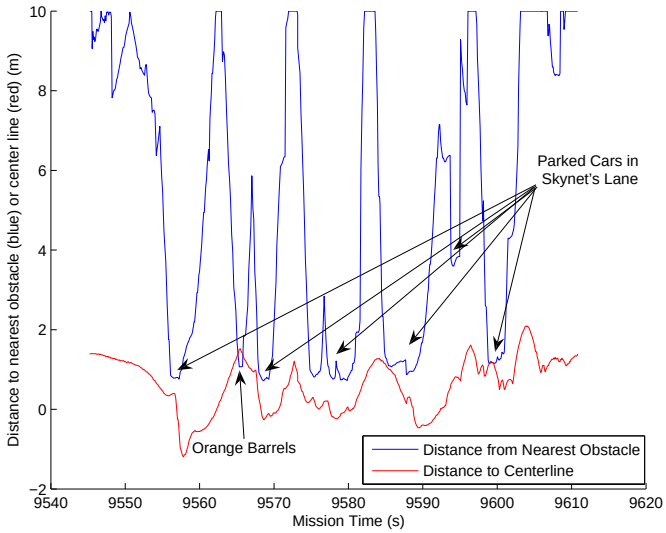


Figure 2.5: Spacing between Skynet and stationary obstacles, and the centerline, during the Gauntlet portion of NQE Area B.

1.4 m to 0.9 m for moving obstacles. These adjustments allowed Skynet to park in parking spaces closely surrounded by obstacles, and made Skynet less sensitive to concrete barriers and other stationary obstacles at the sides of the lanes. Figure 2.5 denotes the sensed spacing between Skynet and other obstacles during the Gauntlet. The lane width in this portion of the course was marked in the RNDF as 12 ft (3.6576 m). Changes to spacing constraints after the first attempt at Area B allowed Skynet to navigate this portion of the course without stopping.

2.4.2 The Wrong Way Vehicle

One event that tested the flexibility of Skynet's Operational Layer occurred early in the UCE, where Skynet encountered and properly dealt with a human-driven Ford Taurus traveling the wrong way on a one-way road. Figure 2.6 (top) shows

the oncoming Taurus from the point of view of the optical cameras and the Operational Layer. Note the Taurus pulled to the left of the lane as far as possible upon passing.

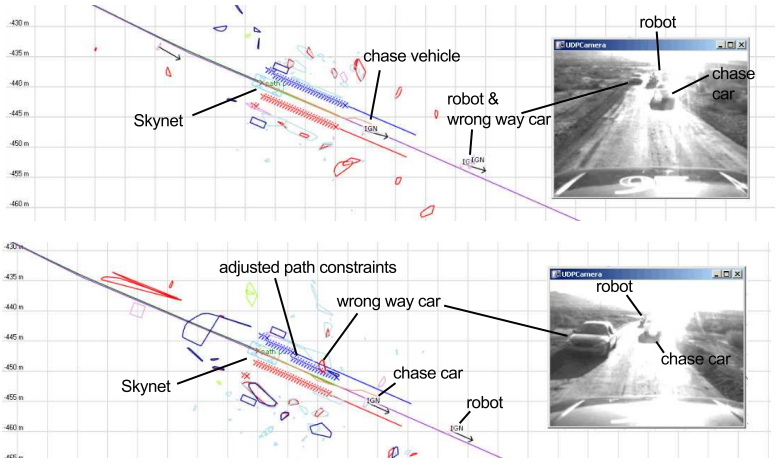


Figure 2.6: (top): Skynet encounters a human-driven vehicle traveling the wrong way on a one-way dirt road. (bottom): The wrong way vehicle is avoided by the Operational Layer, which adjusts path constraints as if the vehicle were a stationary obstacle.

As the wrong way vehicle moved closer to Skynet, the Tactical Planner largely ignored it due to assumptions made in the Tactical Planner. The Tactical Planner assumes all vehicles move in their lanes in the proper direction; therefore, only the closest vehicle in front of Skynet is monitored. As a result, the wrong way vehicle was ignored until it was the closest obstacle to Skynet. At that point the Tactical Planner began to monitor the vehicle, and a fast stop would have been commanded if the vehicle continued to move. Instead, the wrong way vehicle pulled to the side of the road and stopped moving. The Scene Estimator subsequently grouped it with nearby bushes and the dirt berm. As a result, the Operational Layer avoided the wrong way car as a stationary obstacle, as shown in Figure 2.6 (bottom). After successfully avoiding the car, Skynet continued with its mission. This unique incident highlights the Opera-

tional Layer's ability to intelligently handle and avoid unknown obstacles.

2.4.3 Reverse Maneuver

Another event which demonstrated the ability of Skynet's Operational Layer to handle unforeseen circumstances occurred while Skynet was navigating the NQE Area A course. As Skynet was passing a vehicle in the opposing lane, the other vehicle was sensed as being too close, causing Skynet to swerve towards the side of the road to avoid any potential for a collision. The side of the course was lined with concrete barriers; Skynet turned with a large enough angle that when it came to a stop it no longer had enough room between itself and the concrete barrier to turn back into its lane. Figure 2.7 (top) shows Skynet's position when it is stuck against the concrete barrier. Interestingly, this was a scenario very similar to one that caused Cornell's DARPA Grand Challenge robot to cease functioning in 2005 [59].

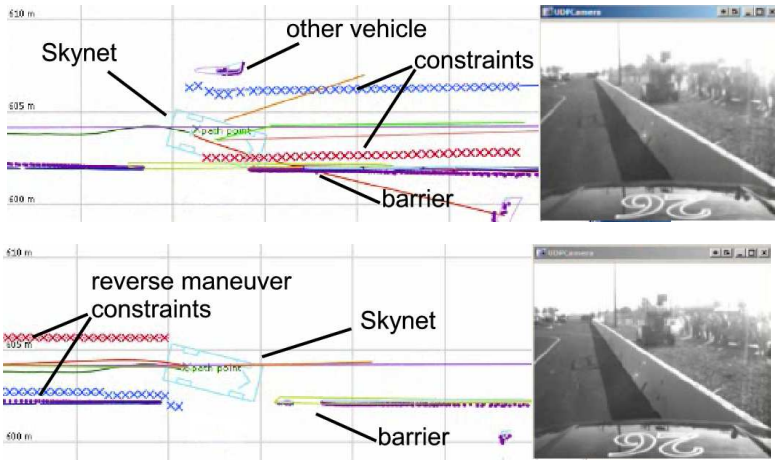


Figure 2.7: (top): Skynet becomes stuck against a concrete barrier on the side of the road. (bottom): Skynet plans a path in reverse to reposition itself back in its lane.

After evaluating the situation, Skynet's Tactical Planner changed Skynet's behavior mode to "reverse" and the Operational Layer planned a smooth, safe path to reverse Skynet back into it's lane. This reverse path is shown in Figure 2.7 (bottom). This incident demonstrated the advantage of having a flexible path planner which could easily be changed to allow for a forward or reverse maneuver. Without this capability, this incident would likely have required a manual reset to free the vehicle.

2.5 Trajectory Optimizer Sensitivity Study

As discussed in Section 2.3, the behavior of the Trajectory Optimizer is governed by two main components: hard constraints as given by Equations 2.5-2.11, and weighted soft constraints given by each component $\alpha_{(\cdot)}[\cdot]$ in Equation 2.1. The hard constraints ensure safe driving behavior by limiting the path based on physical vehicle limitations, safety specifications, and minimum obstacle spacings. The hard constraints do not play a direct role in selecting a best path, but instead serve to eliminate unsafe or unfeasible paths from consideration. The soft constraints are cost penalties in the Trajectory Optimizer's optimization routine which correspond to undesirable driving behaviors such as excessive curvature, low speed, forward and lateral acceleration, and obstacle spacing violation. The Trajectory Optimizer cost function weights, $\alpha_{(\cdot)}$, control the relative importance of each soft constraint. Together, these weighted cost penalties control the nuances of the robot's driving style.

The numerical values of the Trajectory Optimizer weights, $\alpha_{(\cdot)}$, used in the DUC, were tuned manually using both vehicle simulations in the laboratory

and real-time autonomous driving tests in the field [60]. The sensitivity study discussed in this section explores the individual effects of each weight, $\alpha_{(\cdot)}$, on Skynet's overall driving performance using logged data from the DUC semi-final qualification round, the NQE. The NQE portion chosen for the sensitivity analysis is a section of the course known as the 'Gauntlet' [60], because it aggressively exercised the capabilities of the path planner by forcing Skynet to maneuver through a constant stream of parked cars and centerline cones, as seen in Figure 2.8. In Section 2.5.1 the data-driven sensitivity simulation and testing procedure is explained and in Section 2.5.2 the results of the sensitivity analysis are evaluated.

Figure 2.8: Skynet's view of part of the 'Gauntlet' course, with parked cars on both sides of the street and road cones grouped along the centerline

2.5.1 Sensitivity Analysis Procedure

To understand the relationship between Skynet's successful driving behavior and the weights used in its Trajectory Optimizer, a data-driven simulation has been developed to allow a simulated vehicle to navigate the Gauntlet course while experiencing the same sensor data and Tactical Planner instructions that Skynet experienced during the NQE. During Skynet's live autonomous run at the NQE, Scene Estimator data and Tactical Planner instructions were logged at a rate of 10 Hz. This logged data was then played back offline to the Operational Layer and all calls to the Trajectory Optimizer were recorded. The result is a (nominal) 10 Hz log of all the obstacle data flowing into the Trajectory Optimizer, along with the corresponding vehicle position and orientation estimates. This creates a trail of tightly-spaced, experimental obstacle data, which can be transformed into the simulated vehicle's reference frame and used as obstacle information for the simulated vehicle's Trajectory Optimizer. This approach isolates all planning variables above the Trajectory Optimizer, allowing for a consistent, data-driven simulation. The simulation setup is shown in Figure 2.9.

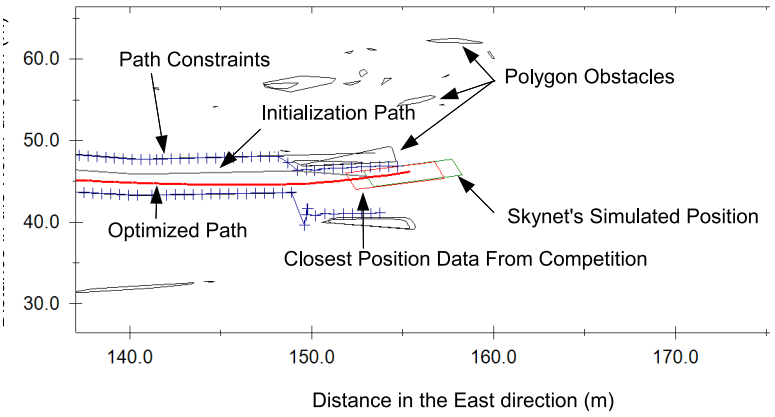


Figure 2.9: Diagram of the sensitivity simulation setup.

Each Trajectory Optimizer weight $\alpha_{(\cdot)}$ is scaled over a range of values by a

scale factor $\beta_{(\cdot)}$. For this study, only one weight is varied at a time. For a given simulation, the weights used are given by:

$$\alpha^j_{(\cdot)} = \alpha^0_{(\cdot)} \cdot \beta^j_{(\cdot)} \tag{2.12}$$

where $\alpha^0_{(\cdot)}$ is the nominal tuned value used during the NQE and j is the index of the current simulation. The nominal values for all of the weighting parameters are given in Table 2.1. The weighting parameter α_w is not included in the sensitivity study because it is only used during specific vehicle maneuvers such as turning at intersections or parking. During these maneuvers, path targets, w_i^{t}, are specified along the base path and the α_w parameter is set significantly higher than the other weighting parameters in order to pull the final path towards the base path. This prevents Skynet from clipping corners. None of these special maneuvers occurred during the Gauntlet.

Table 2.1: Nominal values for Trajectory Optimizer weighting parameters.

α_a^0	α_c^0	α_d^0	α_v^0	$\left[\alpha_q^0\right]_{\text{obstacle}}$	$\left[\alpha_q^0\right]_{\text{lane boundary}}$
10	10	10	10	100	0.1

2.5.2 Sensitivity Analysis Results

All results are presented using two types of figures. The first is a time history plot that shows both the distance to the nearest obstacle metric and the distance to road centerline metric as a function of time. The distance to nearest obstacle metric calculates the Euclidean distance between the closest point on the nearest obstacle and the closest point on the simulated vehicle. The distance to road centerline metric calculates the smallest perpendicular distance to the centerline

of the road, or largest perpendicular distance across the centerline of the road, by any point on the simulated vehicle. All simulations for a given weighting parameter, $\alpha^j_{(\cdot)}$, are plotted together. The nominal value, $\alpha^0_{(\cdot)}$, is shown in black. Variations in $\alpha^j_{(\cdot)}$ are colored according to a gradient scheme, where color intensity indicates the magnitude of the scaling parameter. The second figure is a plot of four performance metrics as a function of simulation scaling factor $\beta^j_{(\cdot)}$. The first two metrics are the extrema of the time history metrics, representing the minimum distance to obstacle and the maximum road centerline violation, respectively, over the entire run. The third metric is the time across centerline, which shows the total accumulated time that any point on the simulated vehicle was across the centerline. The fourth metric is the total run time.

The effects of variations in the weight on acceleration/braking, α_a, are shown in Figure 2.10. The distance to nearest obstacle metric shows a roughly linear relationship, indicating that increasing α_a decreases the vehicle's safety margins around obstacles. Similarly, decreasing α_a increases obstacle safety margins and, up to a certain point, decreases maximum road centerline violations. Typically, these two metrics trade off, but with a low acceleration penalty, the vehicle can slam on its brakes around obstacles in order to improve its performance with respect to both obstacles and the road centerline. Below a scale factor $\beta_a \approx 0.02$, the road centerline performance (largest violation and total violation time) degrades rapidly, as does the vehicle's total time through the course. The total run time plot reveals that this degradation is due to additional evasive maneuvers at low values of α_a (shown in dark colors). These additional evasive maneuvers are likely caused by fluctuations in the obstacle boundaries due to sensor blind spots. With low penalties for high acceleration avoidance maneuvers, minor obstacle variances can cause pronounced vehicle reactions.

The effects of variations in the weight on large curvatures (sharp turns), α_c, are shown in Figure 2.11. The distance to nearest obstacle and distance to road centerline metrics show an inverse relationship. Both have extrema at $\beta_c \approx 0.1$ and then back off and approach a constant value as β_c increases. To the right of these extrema, curvature weighting becomes more significant, reducing the vehicle's ability to maneuver around obstacles. This causes an increase in the distance to road centerline metric and a decrease in the distance to nearest obstacle metric. The total run time metric reveals a linear relation between α_c deviation and the vehicle's total run time through the course. The best performance in terms of obstacle spacing is available at $\beta_c \approx 0.1$, while the best performance in terms of total run time is at high values of β_c, a result of the vehicle being allowed to make shallower turns at the cost of closer obstacle spacing.

The effects of variations in the weight on large changes in curvature (sudden turns), α_d, are shown in Figure 2.12. The distance to nearest obstacle metric and the total run time metric show linear relationships to changes in α_d, and for a majority of the tested range, both can be improved by increasing β_d. Since α_d penalizes *changes* in curvature, this proportional relationship is likely due to α_d forcing a smoother path with less oscillations. The trade-offs to this performance gain are an increase in the time across road centerline metric and a decrease in the distance to road centerline metric for α_d values above the nominal value α_d^0. The vehicle's behavior also becomes more erratic in terms of variations in the metrics at very high values of β_d. However, the nominal value α_d^0 is nearly optimal in selecting the peak performance gains in terms of the distance to nearest obstacle and completion time metrics, without causing a corresponding degradation in performance for the road centerline metric.

The effects of variations in the weight on slow speeds, α_v, are shown in Figure 2.13. For a majority of the tested range, the metric response curves are relatively flat. The likely cause for this result is due to the target velocity output from the Trajectory Optimizer being post processed by the Tracking module. The Tracking module effectively ignores the target velocities provided by the Trajectory Optimizer and replaces them with the largest target velocities possible that still satisfy the vehicle's forward and lateral acceleration constraints. This post processing prevents small velocity weightings in the Trajectory Optimizer from significantly effecting the vehicle's race performance. The velocity weighting α_v is thus limited to primarily affecting the shape of the final path, $p_i \; i \in [1, n]$, since its velocity profile, $v_i \; i \in [1, n]$, is recomputed in the Tracking module. The effect of the speed penalty weight, α_v, is similar to that of the change in curvature weight, α_d, in that both a large β_d and a large β_v favor a smoother final path with fewer oscillations. The beneficial performance influence of α_v breaks down at both the high end and low end of the β_v range tested. For high β_v, the smoothing effect of α_v begins to cause an overdamped response, resulting in sluggish recovery from turns and significant increases in completion time. This overdamped behavior can be seen in the lightly-colored lines in the time history plot of Figure 2.13 (left). For low values of β_v, the final path exhibits significant oscillations which cause a reduction in the distance to nearest obstacle metric and an increase in the completion time metric. This underdamped behavior can be seen in the darkly-colored lines in the time history plot of Figure 2.13 (left).

The effects of variations in the weight on desired obstacle spacing violations, α_q, are shown in Figure 2.14. The metric response curves for most of the range are flat, indicating that α_q overpowers the other cost penalties and prevents any

significant obstacle spacing violations from occurring. As long as the chosen path remains outside of the obstacle desired spacing constraints, the α_q term contributes no penalty and has little effect. For high values of α_q, the metric plots exhibit an erratic behavior, with large spikes in all four metrics. This behavior is likely due to the Trajectory Optimizer attempting to obey potentially conflicting obstacle desired spacing constraints with little regard for the other path performance parameters. This results in pronounced oscillations in regions with many obstacles, as can be seen in the lightly-colored lines in the time history plot of Figure 2.14 (left), especially in the 20-30 second timespan. At the low end of the range there is visible relaxation in the desired spacing constraints. This is indicated by a reduction in the distance to nearest obstacle metric accompanied by a simultaneous improvement in the minimum distance to road centerline and time across centerline metrics.

2.6 Conclusions

This paper presented the derivation, sensitivity study, and implementation of a nonlinear optimization based path planner for large scale robots operating in dynamic urban environments. The path planner was experimentally implemented on Skynet, an autonomous robot built by Cornell University for the 2007 DARPA Urban Challenge. Skynet's Operational Layer utilizes a novel mixture of a discrete grid based search algorithm to identify a drivable corridor and a constrained nonlinear optimization routine to generate a smooth, predictable, drivable path. The hard constraints used in the optimization guarantee that the final path never violates minimum obstacle spacings, and that the path obeys safety and vehicle limitations. The soft constraints used in the optimization are

cost penalties which correspond to undesirable driving behaviors such as excessive curvature, speed, acceleration, and obstacle spacing violation.

Skynet's performance in the DUC demonstrated the robustness of this planning approach and its ability to handle unforeseen circumstances. Skynet's path planner enabled it to navigate a narrow, obstacle filled section known as the "Guantlet," to avoid a car driving the wrong way down a one way road, and to seamlessly plan paths in reverse.

The sensitivity study presented in this paper provides intuition about the behavioral dependence of the Operational Layer's Trajectory Optimizer on the optimization weights. The sensitivity study is performed using a data-driven simulation based on logged data from the DUC. While most of the weights have predictable influences for a majority of their tested range, it is clear that too little or too much of any one weight causes an undesirable effect in one or more performance metrics. There are also several surprises, such as speed weight, α_v, having minimal influence due to post-processing of the speed profile, and the obstacle spacing weight, α_q, having an effect only at the extreme ends of its range because it is large enough to prevent any significant obstacle spacing violations from occurring.

2.7 Funding

This work was supported by the DARPA Urban Challenge program (contract no. HR0011-06-C-0147), with Dr. Norman Whitaker as Program Manager.

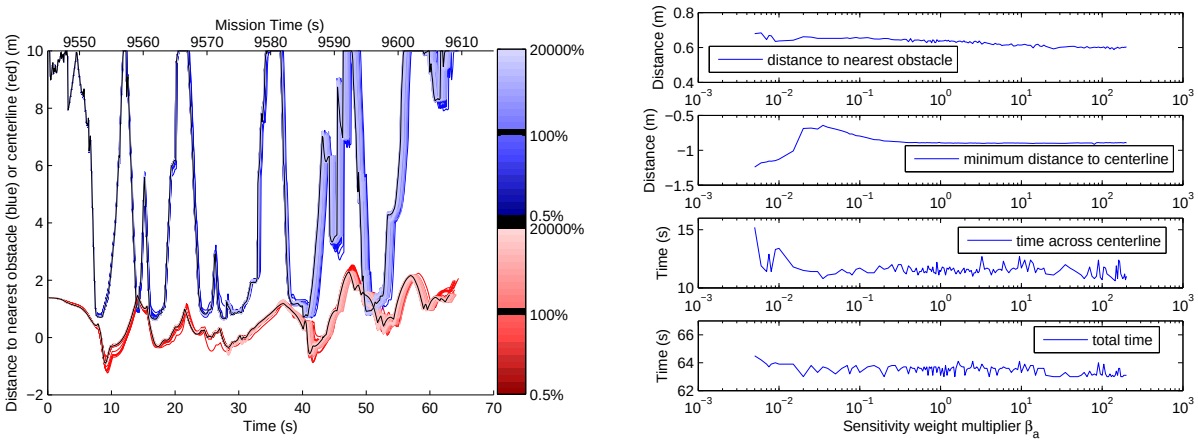


Figure 2.10: Sensitivity results for forward acceleration weighting α_a. (Left): A time history plot showing distance to nearest obstacle and distance to road centerline as a function of time. (Right): Four plots of performance metrics as a function of simulation weight multiplier β_a.

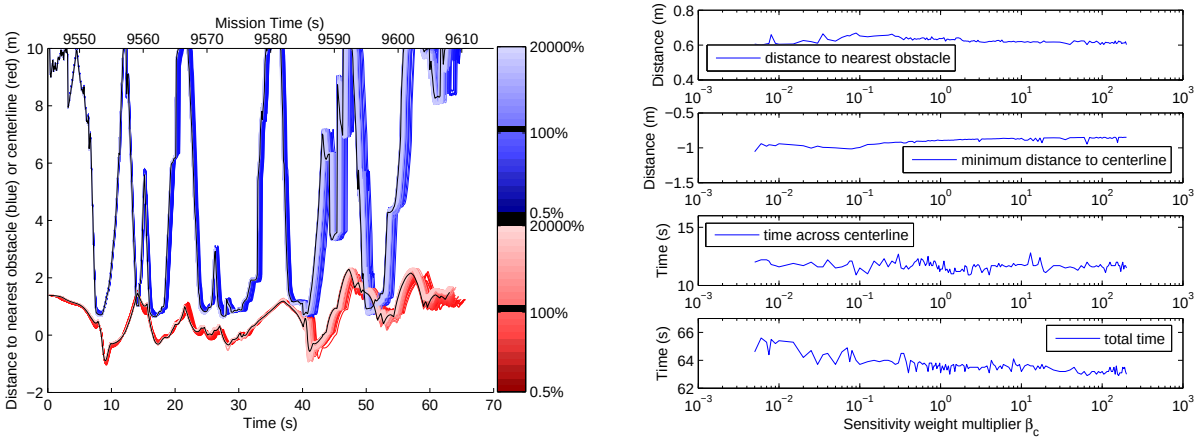


Figure 2.11: Sensitivity results for curvature weighting α_c. (Left): A time history plot showing distance to nearest obstacle and distance to road centerline as a function of time. (Right): Four plots of performance metrics as a function of simulation weight multiplier β_c.

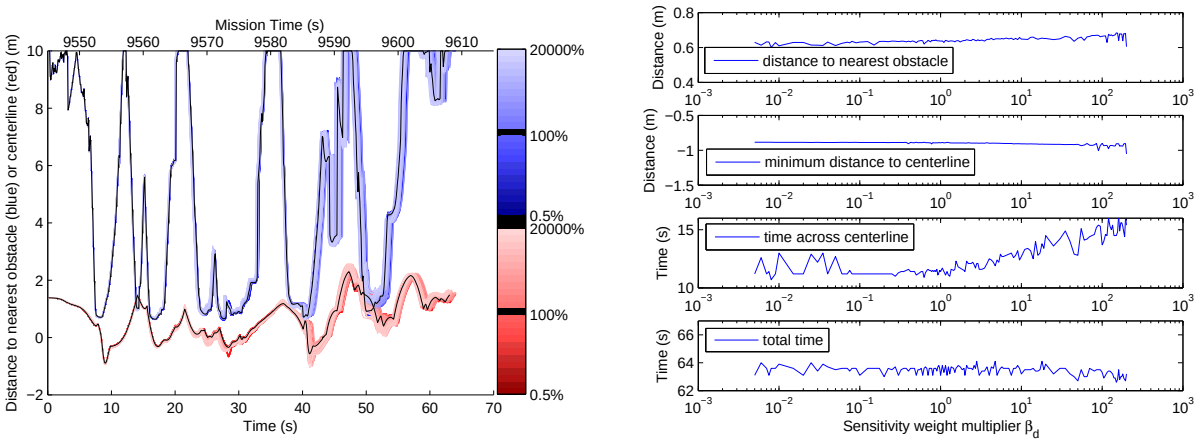


Figure 2.12: Sensitivity results for differential curvature weighting α_d. (Left): A time history plot showing distance to nearest obstacle and distance to road centerline as a function of time. (Right): Four plots of performance metrics as a function of simulation weight multiplier β_d.

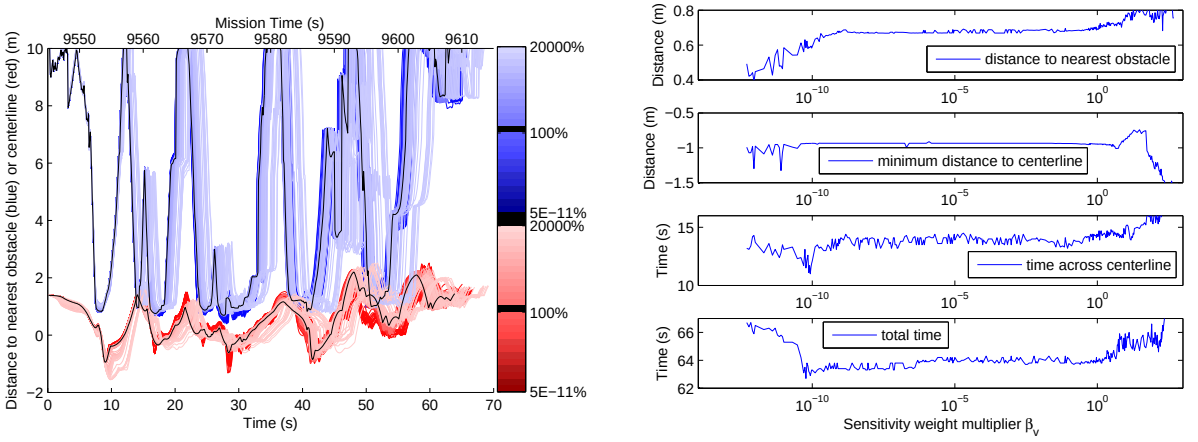


Figure 2.13: Sensitivity results for velocity weighting α_v. (Left): A time history plot showing distance to nearest obstacle and distance to road centerline as a function of time. (Right): Four plots of performance metrics as a function of simulation weight multiplier β_v.

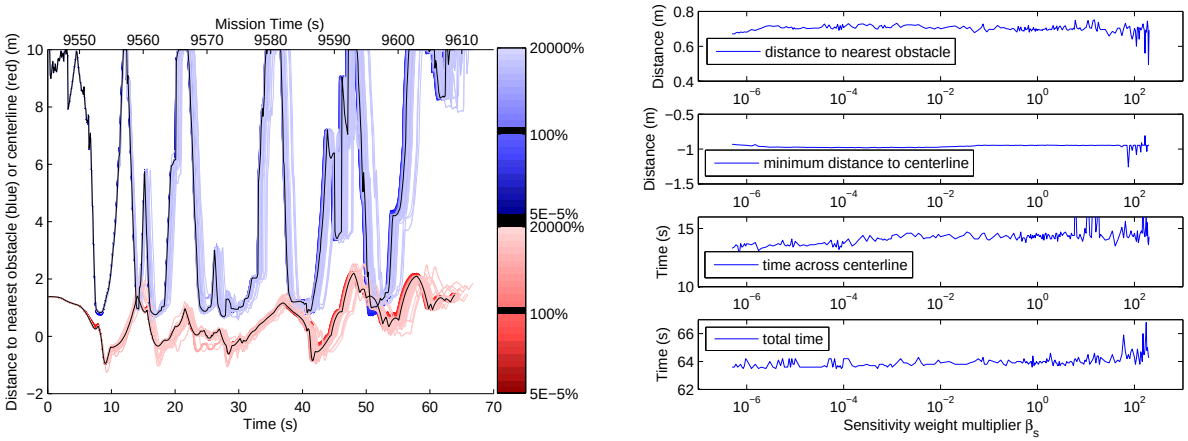


Figure 2.14: Sensitivity results for obstacle desired spacing violation weighting α_q. (Left): A time history plot showing distance to nearest obstacle and distance to road centerline as a function of time. (Right): Four plots of performance metrics as a function of simulation weight multiplier β_q.

CHAPTER 3

CONTINGENCY PLANNING OVER PROBABILISTIC OBSTACLE PREDICTIONS FOR AUTONOMOUS ROAD VEHICLES

Abstract

This paper presents a novel optimization based path planner capable of planning multiple contingency paths to directly account for uncertainties in the future trajectories of dynamic obstacles. This planner addresses the particular problem of probabilistic collision avoidance for autonomous road vehicles that are required to safely interact, in close proximity, with other vehicles with unknown intentions. The presented path planner utilizes an efficient spline based trajectory representation and fast but accurate collision probability bounds to simultaneously optimize multiple continuous contingency paths in real-time. These collision probability bounds are efficient enough for real-time evaluation, yet accurate enough to allow for practical close-proximity driving behaviors such as passing an obstacle vehicle in an adjacent lane. An obstacle trajectory clustering algorithm is also presented to enable the path planner to scale to multiple-obstacle scenarios. Simulation results show that the contingency planner allows for a more aggressive driving style than non-contingency based approaches without compromising the overall safety of the robot.

3.1 Introduction

Current autonomous vehicles are adept at identifying obstacles, planning routes, and interacting in controlled environments with well defined rules, as demonstrated in the 2007 DARPA Urban Challenge (DUC) [10]. The DUC also demonstrated that there remains a large gap in problem solving and decision making capabilities between autonomous vehicles and human drivers. A major reason for this gap is the inability of current algorithms to adequately identify, predict, and utilize obstacle intent [20]. Inferring obstacle intent, such as intended obstacle goals, enables an autonomous vehicle to anticipate the future motion of dynamic obstacles, which can then be used to improve the safety and robustness of the robot's motion planning.

Previous research in this area has contributed potential solutions to two primary parts of the dynamic obstacle avoidance problem: 1) predicting future obstacle motion and 2) path planning in dynamic environments using predicted obstacle motions. Predicting future obstacle motion is a problem that has been explored in a variety of fields and contexts. Kushleyev and Likhachev [47] perform predictions of dynamic obstacles using constant velocity and constant curvature dynamics models. This approach is computationally efficient, but ignores potentially useful structural information in the environment. Miura and Shirai [63] infer the potential paths an obstacle might take using a tangent graph of the environment and a 1D model of velocity variance along each potential path. This approach utilizes structural information in the environment, but oversimplifies obstacle motion and forms overconfident predictions. Hwang and Seah [37] present an algorithm for inferring the intent of other airplanes based on likely flight plans, and for probabilistically modeling their future mo-

tion using bounded process noise and probabilistic models of transitions between maneuvers. This approach is useful for air vehicles but not readily adaptable for road vehicles which have fewer preset maneuvers and much more restrictive environmental constraints.

Non-parametric learning methods have also been applied to obstacle prediction. Tay and Laugier [75] use Gaussian Processes to model multiple possible future trajectories based on observed training data. Joseph et al. [40] and Kim, Lee, and Essa [44] both use Gaussian Processes to model trajectory derivatives for road vehicles. Bennewitz et al. [6] cluster observed trajectories of human participants through an environment into nominal trajectories and develop a Hidden Markov Model to predict mode transitions. These non-parametric approaches can produce detailed probabilistic trajectories but they generally require retraining for each new environment and obstacle type.

A variety of path planning strategies have been developed to incorporate dynamic obstacle predictions. Dynamic obstacles present a significant planning challenge because they increase the dimensionality of the planning problem. Simple dynamic planning approaches involve reducing dimensionality by projecting the predicted obstacle trajectories onto the static environment or decoupling the problem into a static motion planning problem and a one dimensional dynamic velocity planning problem [18]. Approaches which attempt to solve the full dimensional dynamic planning problem typically avoid directly optimizing over the continuous search space through the use of probabilistic sampling or through discretization of the search space. Aoude et al. [4] use a predicted tree of reachable trajectories for each dynamic obstacle to bias a rapidly-exploring random tree based path planner. Kushleyev and Likhachev [47] use a

time-dependent variation of A* which searches over a set of predefined motion primitives to avoid collisions with dynamic obstacles. These discrete and sampling based approaches provide only an approximate optimal path due to discretization error and limited sample size. Additionally, these algorithms ignore the fact that multiple motion predictions for a single obstacle, due to uncertainties in the obstacle's intent, are mutually exclusive events.

Obstacle prediction has also been applied directly to the problem of autonomous vehicle navigation. During the DUC, the Tartan Racing team showed that it is possible to deduce a small set of short term goal hypotheses for dynamic obstacles based on the surrounding road configuration [17]. The possible goals for another vehicle approaching a four way intersection, for example, would be to stop, continue straight, turn left, or turn right. These inferred goal hypotheses are then used to make deterministic predictions about each dynamic obstacle's possible future trajectory. The simulations presented in [17] and the success of Tartan Racing's BOSS robot in the DUC [78] make a compelling case for the inclusion of obstacle intent in the planning process. However, the use of deterministic motion predictions in [17] can make the collision probability assumptions of the robot overconfident.

To provide better dynamic planning capabilities for autonomous road vehicles, a path planning formulation is presented which incorporates probabilistic obstacle predictions to enable efficient generation of a safe set of contingency paths in dynamic environments. Goal hypotheses are formed for each dynamic obstacle based on the topology of the road network, and the obstacles' state distributions are predicted forward in time using a probabilistic motion model. Path optimization is based on a nonlinear, constrained, numerical

optimization formulation; extending the work in [60], [33]. Formulating the path planning problem as a constrained numerical optimization problem is a well studied path optimization technique. Full nonlinear optimization is used by Milam, Mushambi, and Murray [55] who generate nonlinear trajectories using sequential quadratic programming, while linear and quadratic path optimization formulations have been developed and implemented by Blackmore, Li, and Williams [7] and Bellingham, Richards, and How [5]. A key advantage of the numerical optimization formulation in this paper is its computational efficiency, enabling the dynamic planning problem to be solved in real-time. This efficiency is achieved by 1) adopting an efficient spline based trajectory representation, 2) solving the planning problem over a limited planning horizon, and 3) using fast but accurate collision probability bounds. While limiting the planning horizon prevents true global planning, it has been shown to be an effective navigation strategy when used in conjunction with a higher level global route planner [60].

The key novel contributions of this paper include 1) the simultaneous optimization of partially shared contingency trajectories over the robot's continuous planning space in order to accurately handle mutually exclusive obstacle predictions, 2) a computationally efficient and accurate upper bound on pointwise collision probability between two polygonal objects with uncertain relative position and orientation, 3) a spline based trajectory representation and associated cost function and constraint definitions for expressing the contingency path planning problem using only a small number of optimization variables, and 4) an obstacle trajectory clustering algorithm designed to reduce planning complexity by capturing the most important mutually exclusive obstacle decisions using a limited number of obstacle trajectory clusters. Spline based trajectory

representations have been used before, such as in [13]; here cubic splines are employed in a unique way to enable multiple contingency trajectories to be efficiently optimized in unison. The contingency planning approach presented in this paper was introduced in [30], and the obstacle trajectory clustering algorithm was introduced in [31]. This paper includes new simulation results and performance analysis, a sensitivity analysis of the collision probability threshold constraint, and expanded analysis of the collision probability bound.

This paper is arranged as follows: Section 3.2 introduces the path optimization framework and proposes an algorithm for probabilistically predicting the motion of dynamic obstacles; Section 3.3 presents the spline based path representation, the path optimization algorithm, and the collision probability bound methods; Section 3.4 presents a method for clustering obstacle trajectory predictions to improve scaling; Section 3.5 present simulation results and Section 3.6 provides conclusions.

3.2 Contingency Planning Architecture and Obstacle Prediction

The system architecture for the proposed Contingency Planner is shown in Figure 3.1. In this architecture, the Contingency Planning procedure consists of an Obstacle Prediction phase followed by a Path Optimization phase. The Contingency Planning approach presented in this paper assumes that the problems of identifying unique dynamic obstacles, tracking these obstacles between sensor measurements, and estimating the state of these obstacles have been resolved to a sufficient degree using an external tracking algorithm; Ref. [61] is one ex-

ample of such an algorithm. As such, current Gaussian obstacle state estimates, defined by their mean and covariance, (μ_i, Σ_i) for the i^{th} of n_o dynamic obstacles, are assumed to be known from the Obstacle Tracker at the beginning of each Contingency Planning cycle.

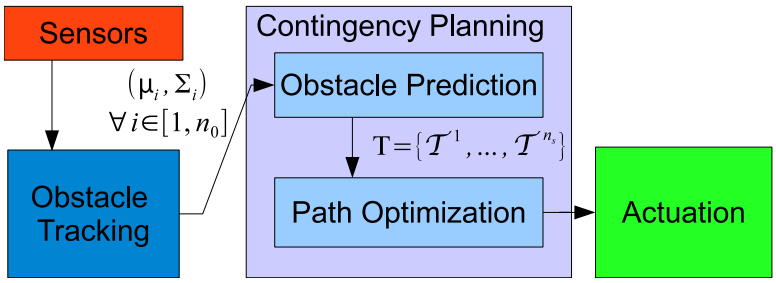


Figure 3.1: Block diagram of the contingency planning architecture.

At the start of the Obstacle Prediction phase, a set of discrete goal modes, $\mathcal{G}_i$, is inferred for each obstacle based on a known topological representation of the local road network and the planning horizon of the robot. For example, a dynamic obstacle approaching a four way intersection might use $\mathcal{G} = \{\text{stop}, \text{go straight}, \text{turn left}, \text{turn right}\}$. This concept of goal modes is very general and can be extended to encode other sources of obstacle uncertainty, such as including multiple motion models to represent 'good,' 'aggressive,' or 'impaired' drivers or to represent obstacle classification uncertainty such as 'car' vs. 'bicycle'.

When a new obstacle is observed, a new set of goal modes is inferred based on the topology of the surrounding road network. For this paper, all possible goal modes for a newly observed obstacle are initialized with equal probability. However, *a priori* information such as whether the obstacle is in a turn lane, along with any observed turn signals or hand signs, can also be used to more intelligently initialize these goal mode probabilities. Once a goal mode becomes

infeasible, it is dropped and the probabilities for the other goal modes are renormalized. Similarly, when a tracked obstacle prediction reaches a new decision point, such as a new intersection, its goal mode is split into a set of new goal modes based on the road network topology.

The state distributions for each obstacle are predicted forward in time for each possible goal mode using a probabilistic motion model. The contingency planning approach proposed here is general to a wide variety of obstacle prediction models. For example, the motion model can vary in complexity from a simple path following model that tracks the center of the driving lane, to individual path planners designed to model the motion of each obstacle type. For linear motion models, this prediction can be performed using the prediction step of the standard Kalman filter to yield the desired predicted mean and covariance at each timestep. For more general nonlinear, non-differentiable motion models, this prediction can be performed using an algorithm such as the Sigma Point Transform [42].

Formally, let $\mathcal{T}$ represent a set of predicted obstacle trajectories. The set of trajectories $\mathcal{T}_i$ represents all the predicted trajectories for the i^{th} obstacle corresponding to the goal modes in $\mathcal{G}_i$. Each predicted trajectory for obstacle i, $T \in \mathcal{T}_i$, is represented by a sequence of obstacle state distributions over N timesteps into the future:

$$T = \{(\mu_i, \Sigma_i), (\hat{\mu}_i, \hat{\Sigma}_i)_1, \ldots, (\hat{\mu}_i, \hat{\Sigma}_i)_N\} \tag{3.1}$$

where (μ_i, Σ_i) represents the current state estimate of obstacle i and $(\hat{\mu}_i, \hat{\Sigma}_i)_k$ represents a predicted obstacle state distribution at future timestep k.

For multiple dynamic obstacles, the set of all possible permutations of pre-

dicted obstacle trajectories is defined as:

$$\mathbf{T} = \mathcal{T}_1 \times \mathcal{T}_2 \times \ldots \times \mathcal{T}_{n_o} \tag{3.2}$$

where $\mathbf{T}$ is the Cartesian product of the predicted obstacle trajectory sets for all n_o obstacles. For a two obstacle environment with $\mathcal{T}_1 = \{T_\mathrm{a}, T_\mathrm{b}\}$ and $\mathcal{T}_2 = \{T_\mathrm{c}, T_\mathrm{d}\}$, $\mathbf{T} = \mathcal{T}_1 \times \mathcal{T}_2 = \{\{T_\mathrm{a}, T_\mathrm{c}\}, \{T_\mathrm{a}, T_\mathrm{d}\}, \{T_\mathrm{b}, T_\mathrm{c}\}, \{T_\mathrm{b}, T_\mathrm{d}\}\}$. The set of trajectories $\mathcal{T}^j \in \mathbf{T}$ is used to represent the j^{th} set of trajectory permutations, such that $\mathbf{T} = \{\mathcal{T}^1, \ldots, \mathcal{T}^{n_s}\}$ where $n_s = |\mathbf{T}|$ represents the cardinality of $\mathbf{T}$, defined as:

$$n_s = \prod_{i=1}^{n_o} |\mathcal{T}_i| \tag{3.3}$$

At the start of each new Contingency Planning cycle, updated state estimates for each obstacle, $\{(\mu_1, \Sigma_1)_1, \ldots, (\mu_{n_o}, \Sigma_{n_o})_1\}$, are obtained from the Obstacle Tracker, as shown in Figure 3.1. These new state estimates are used to update the probability of each obstacle trajectory permutation, $p(\mathcal{T}^j)$:

$$p(\mathcal{T}^j)_1 = \frac{p(\mathcal{T}^j) \prod_{i=1}^{n_o} p((\mu_i, \Sigma_i)_1 | (\hat{\mu}_i, \hat{\Sigma}_i)_1^j)}{\sum_{\mathcal{T}^m \in \mathbf{T}} p(\mathcal{T}^m) \prod_{i=1}^{n_o} p((\mu_i, \Sigma_i)_1 | (\hat{\mu}_i, \hat{\Sigma}_i)_1^m)} \tag{3.4}$$

where $p((\mu_i, \Sigma_i)_1 | (\hat{\mu}_i, \hat{\Sigma}_i)_1^j)$ is the innovation likelihood of the updated state estimate for obstacle i, $(\mu_i, \Sigma_i)_1$, given the predicted state distribution from the previous timestep, $(\hat{\mu}_i, \hat{\Sigma}_i)_1^j$ from the trajectory prediction for obstacle i included in $\mathcal{T}^j$.

3.3 Path Optimization

The path planning problem is formulated here as a general nonlinear constrained optimization problem:

$$\mathbf{h}_{\text{opt}} = \arg\min_{\mathbf{h}} \; J(\mathbf{h}) \qquad C(\mathbf{h}) < 0 \tag{3.5}$$

where $\mathbf{h}$ is a parameter vector that defines a trajectory or set of contingency trajectories for the robot and $J(\mathbf{h})$ is a predefined cost function over which the path is optimized. The constraint function, $C(\mathbf{h})$, places constraints on the range of possible values that $\mathbf{h}$ can take. With this formulation, the path planning problem can be solved using well known convex optimization techniques, including interior point methods [21] and sequential quadratic programming methods [8]. For non-convex cost function and constraint definitions, this approach risks convergence of the optimization to a suboptimal local minimum; however, feasible local minima still represent acceptable, drivable paths. The key advantage of this approach over contemporary heuristic path planning approaches is that first and second derivative information from the cost function, $J(\mathbf{h})$, and constraint function, $C(\mathbf{h})$, is used to guide the path optimization search.

3.3.1 Contingency Planning Approach

The goal of contingency planning is to generate a set of planned paths that account for all possible evolutions of the robot's environment. Here, uncertainty in the future state evolution of the robot's environment is encoded as a set of mutually exclusive obstacle predictions, generated by the prediction ap-

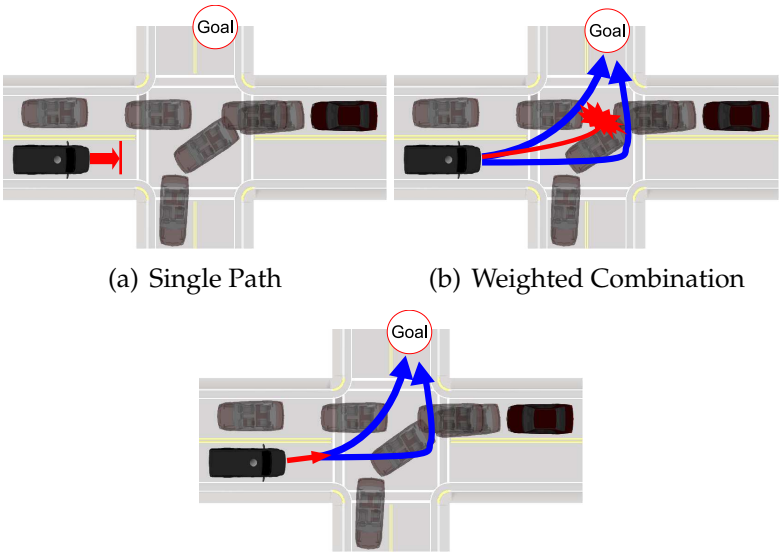
proach outlined in Section 3.2. In this framework, a separate contingency path is planned for each possible permutation of predicted obstacle trajectories, $\mathcal{T}^j$. The approach used here further constrains all contingency paths to share the same initial segment. By using a single shared path segment at the start of the path, the Contingency Planner can provide the robot with an immediate deterministic action; multiple distinct contingencies for the rest of the path are then used to account for uncertainty in the evolution of the robot's dynamic environment.

This partially shared contingency planning approach is preferable to alternate approaches, such as planning a single path which attempts to avoid all obstacle predictions simultaneously as shown in Figure 3.2(a), or taking a weighted combination of independent contingency paths as shown in Figure 3.2(b). Planning a single path is overcautious and ignores the fact that the obstacle goal sets are mutually exclusive, while taking a weighted combination of independent contingency paths offers no guarantees on the safety of the combined path.

Sharing the initial segment of each contingency path, as shown in Figure 3.2(c), gives the robot a deterministic action to execute at the current time step while maintaining the independence of future contingencies. In this contingency planning formulation, the path optimization cost function in Equation 3.5 can be expanded as shown in Equation 3.6:

$$J(\mathbf{h}) = J_{\text{shared}}(\mathbf{h}, \mathbf{T}) + \sum_{\mathcal{T}^j \in \mathbf{T}} p(\mathcal{T}^j) J_{\text{conting}}(\mathbf{h}, \mathcal{T}^j) \tag{3.6}$$

where $J_{\text{shared}}(\mathbf{h}, \mathbf{T})$ is the cost for the shared path segment, and $J_{\text{conting}}(\mathbf{h}, \mathcal{T}^j)$ is the cost for the j^{th} contingency path.



(a) Single Path (b) Weighted Combination

(c) Shared Segment

Figure 3.2: A comparison of different strategies for path planning using mutually exclusive sets of obstacle predictions.

Both $J_{\text{shared}}(\mathbf{h}, \mathbf{T})$ and $J_{\text{conting}}(\mathbf{h}, \mathcal{T}^j)$ in equation 3.6 are formulated as a summation of weighted terms designed to penalize undesirable driving behaviors; terms include penalty functions on large changes in vehicle dynamics, on proximity to static and dynamic obstacles, and on the distance between the robot and its goal. Equations 3.7 and 3.8 show the component cost terms in J_{shared} and J_{conting} respectfully:

$$J_{\text{shared}} = J^{\text{dynamics}}_{\text{shared}} + J^{\text{static}}_{\text{shared}} + J^{\text{collision}}_{\text{shared}} \tag{3.7}$$

$$J_{\text{conting}} = J^{\text{dynamics}}_{\text{conting}} + J^{\text{static}}_{\text{conting}} + J^{\text{collision}}_{\text{conting}} + J^{\text{goal}}_{\text{conting}} \tag{3.8}$$

where $J^{\text{dynamics}}_{(\cdot)}$ contains terms penalizing large accelerations and large path curvatures, $J^{\text{static}}_{(\cdot)}$ penalizes proximity to static obstacles, $J^{\text{collision}}_{(\cdot)}$ penalizes high dynamic obstacle collision probabilities, and $J^{\text{goal}}_{(\cdot)}$ penalizes the distance between the end of a robot's planned path and its goal.

The rest of Section 3.3 presents the path parameterization and the path optimization cost terms and constraints in detail. Section 3.3.2 presents a novel spline based path parameterization that efficiently encodes multiple shared contingency paths. Section 3.3.3 outlines a static obstacle penalty term, f_{prx}, which is used in the $J^{\text{static}}_{(\cdot)}$ cost terms. Section 3.3.4 presents a novel algorithm for computing a tight bound on the point-wise collision probability between two polygons with uncertain relative orientation and position. This collision probability bound is used as the basis for a dynamic obstacle collision probability optimization constraint. Section 3.3.5 presents a full summary of the path optimization algorithm, including a full description of the objective function terms and all of the constraint functions.

3.3.2 Trajectory Representation

A key challenge when planning multiple contingency paths in real-time is defining an efficient trajectory representation. Such a trajectory representation must be both expressive in its ability to cover the search space and compact in its dimensionality. Cubic splines are chosen for this purpose because two splines, $y(t)$ and $x(t)$, define a continuous representation of the robot position, (x, y), and its derivatives, (v^x, v^y, a^x, a^y), as a function of time, t, using only a small set of control points, h, as variables. The optimization vector, $\mathbf{h}$, is a set of parameters defining n_s contingency trajectories with a shared initial segment:

$$\mathbf{h} = \{h^x_1, h^y_1 | h^{1,x}_{2:n}, h^{1,y}_{2:n}, ..., h^{n_s,x}_{2:n}, h^{n_s,y}_{2:n}\} \tag{3.9}$$

where n is the number of cubic line segments in each contingency path. The shared initial segment constraint is enforced by defining the initial segment as

an independent cubic line segment, parameterized by the robot's current state and the first control point, (h_1^x, h_1^y). The first control point, (h_1^x, h_1^y), represents the end of the initial shared segment, and a separate cubic spline is defined for each contingency path starting from this point, $\{h_{2:n}^{1,x}, h_{2:n}^{1,y}, ..., h_{2:n}^{n_s,x}, h_{2:n}^{n_s,y}\}$.

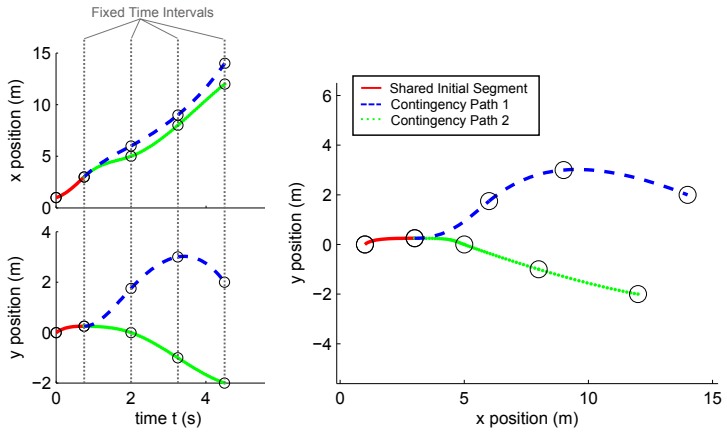


Figure 3.3: Depictions of the spline based path representations as a function of time (left), and in physical space (right).

Figure 3.3 depicts a cubic spline representation for two contingency paths. In order to reduce dimensionality, the control points on the spline are constrained to lie on fixed time intervals, as indicated by the dashed vertical lines. By using this cubic spline definition, the robot's position and its derivatives at a given time t depend linearly on the control point parameters $\mathbf{h}$.

The optimization vector $\mathbf{h}$ is initialized by setting all contingency paths equal to a nominal path. The nominal path is based on the robot following the lane center in an obstacle free environment.

3.3.3 Static Obstacle Cost Map

To handle static obstacles and static driving boundaries such as road lane boundaries, a driving corridor is defined that explicitly expresses an obstacle free space that the robot is expected to drive through. Driving corridor descriptions are common in mobile robotics including Geraerts and Overmars [25] and Wein, Van Den Berg and Halperin [84]. Driving corridor cost terms have also been applied in the context of autonomous vehicles, most notably by Dolgov et al. [14], who develop a local minima free cost map based on a Voronoi decomposition of the robot's static obstacle map. The driving corridor definition presented here is designed to provide an easily differentiable static obstacle cost term that is convex with respect to the lateral offset of the robot from the center of the driving corridor.

The obstacle free driving corridor is represented by a sequence of connected line segments defining the centerline of the corridor along with an associated width for each centerline vertex node. For driving in unstructured environments, an obstacle free driving corridor is identified using a discrete search over a grid based representation of the environment, as in [33]. For structured environments, such as driving on a road network, an obstacle free driving corridor is identified using a combination of *a priori* information, such as road lane definitions, and sensed obstacle information. In the lane driving scenario, the centerline and node width information is obtained trivially from the lane definitions in the *a priori* map. For the unstructured environment scenario, or when the lane centerline must be shifted due to static obstacles such as parked cars, a centerline can be obtained using a straight skeleton decomposition of the corridor, as in [1]. Centerline node widths can then be computed by solving for the

minimum distance to the boundary for each node.

A penalty function is defined in the path planning optimization (as part of $J^{\text{static}}_{(\cdot)}$ in Equations 3.7,3.8) to bias the robot away from static obstacles in the environment. This penalty function, $f_{\text{prx}}(x, y)$, is defined as a normalized distance between an evaluation point along the robot's path, (x, y), and the centerline of the driving corridor:

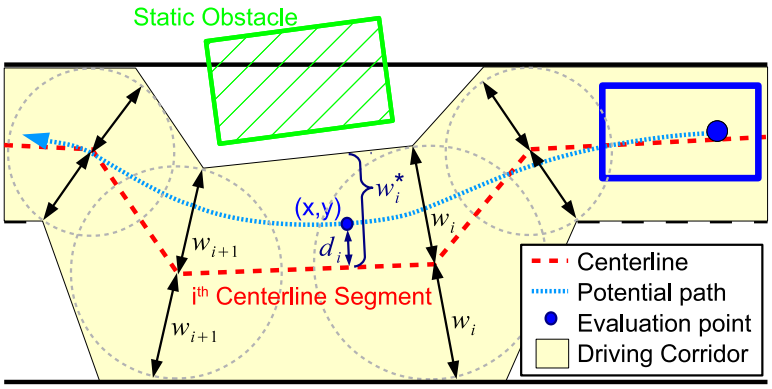


Figure 3.4: Diagram of the distance calculation between an evaluation point along the robot's path and a single segment of the centerline.

$$f_{\text{prx}}(x, y) = \min_i \left(\frac{d_i}{w_i^*} \right)^2 \quad \forall i \in [1, n_c] \tag{3.10}$$

where d_i is the minimum distance between the robot position (x, y) and the i^{th} centerline segment, w_i^* is the interpolated width of the driving corridor for the i^{th} centerline segment at the robot position, and n_c is the total number of centerline segments. Figure 3.4 shows this calculation for the i^{th} centerline segment. This quadratic cost function avoids local minima by having a zero cost centerline, is convex with respect to the robot's lateral offset in its driving lane, and allows for straightforward calculation of derivatives even within obstacle

regions.

3.3.4 Collision Probability

The problem of calculating collision probabilities with uncertain dynamic obstacles has been studied in a range of fields including air traffic control [65], satellite collision avoidance [2], and mobile robotics [49]. Figure 3.5 depicts a typical sce-

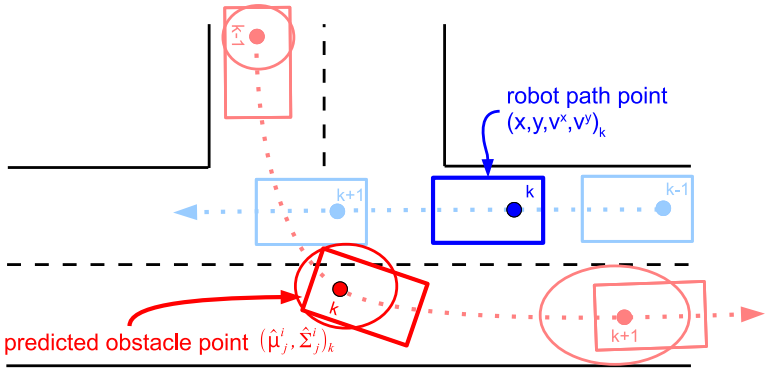


Figure 3.5: Diagram of a scenario in which the dynamic collision probability calculation must be performed for multiple discrete timesteps.

nario where point-wise collision probability calculations must be computed for multiple timesteps; timestep k is highlighted. For notational simplicity in the subsequent collision probability derivation, the timestep k case is considered and the k notation is dropped for now. Let $z_{\text{robot}} = [x_{\text{robot}}, y_{\text{robot}}]$ represent the interpolated position of the robot, and let $\mu^z_{\text{obst}} = [\mu^x_{\text{obst}}, \mu^y_{\text{obst}}]$ and Σ^z_{obst} represent the mean position and 2×2 position error covariance of the obstacle. A normal distribution is assumed for the obstacle position distribution, as this is consistent with the assumptions of the Obstacle Tracker and the Obstacle Prediction algorithm presented in Section 3.2. More general obstacle position distributions

can be accommodated using a mixture of Gaussians representation.

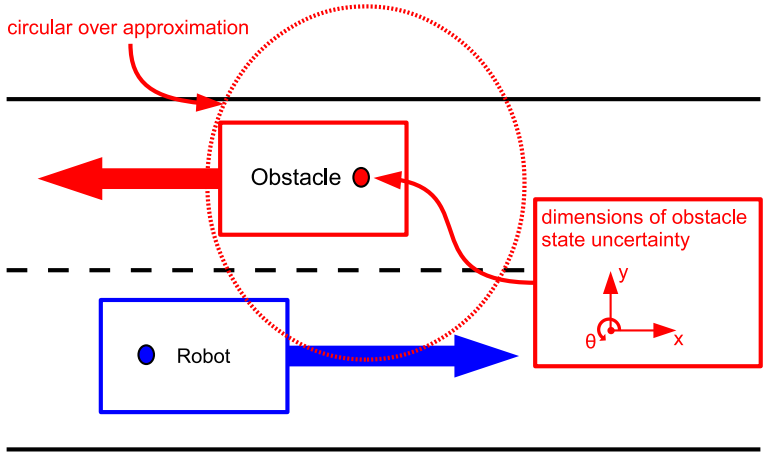


Figure 3.6: Depiction of a circular over-approximation for the shape of a road vehicle in an adjacent lane.

For the application of path planning in autonomous road vehicles, the collision probability calculation is especially challenging because road vehicles are expected to interact closely with dynamic obstacles. This prevents the use of circular over-approximations of the obstacle shape to account for uncertainties in obstacle's orientation. Figure 3.6 shows a common scenario in which the conservativeness of a circular collision probability bound is a problem: encountering a vehicle traveling in the opposite direction on a two lane road. In this case, using a circular collision probability bound, would prevent basic, expected driving behavior, such as driving past the obstacle vehicle in an adjacent lane.

A tighter upper bound can be achieved by using the actual rectangular robot and obstacle shapes. This complicates the collision probability calculation because the relative obstacle orientation becomes an additional uncertain variable. For computational simplicity, the obstacle orientation is assumed to be an independent 1D Gaussian distribution, $\gamma_{\text{obst}} \sim \mathcal{N}(\mu^{\gamma}_{\text{obst}}, (\sigma^{\gamma}_{\text{obst}})^2)$; this is equivalent to

ignoring correlations between the orientation and position elements of the obstacle's state error covariance. Using this assumption, the collision probability can be found by marginalizing over the obstacle orientation uncertainty:

$$p_{\text{coll}}(z_{\text{robot}}, \mu^z_{\text{obst}}, \Sigma^z_{\text{obst}}, \mu^\gamma_{\text{obst}}, \sigma^\gamma_{\text{obst}}) = \int_{\gamma_{\text{obst}}} p(\gamma_{\text{obst}}) p^{\text{poly}}_{\text{coll}}(z_{\text{robot}}, \mu^z_{\text{obst}}, \Sigma^z_{\text{obst}}, \gamma_{\text{obst}}) \tag{3.11}$$

where $p^{\text{poly}}_{\text{coll}}(z_{\text{robot}}, \mu^z_{\text{obst}}, \Sigma^z_{\text{obst}}, \gamma_{\text{obst}})$ is the point-wise collision probability between the robot and the obstacle for a fixed relative orientation γ_{obst}. An upper bound on the integral in Equation 3.11 can be found by taking a sum over n_γ discrete obstacle orientation ranges, $[\gamma^{l-1}_{\text{obst}}, \gamma^l_{\text{obst}}]$ for $l \in [1, n_\gamma]$, where $\sum_{l=1}^{n_\gamma} p([\gamma^{l-1}_{\text{obst}}, \gamma^l_{\text{obst}}]) = 1$.

$$p_{\text{coll}}(z_{\text{robot}}, \mu^z_{\text{obst}}, \Sigma^z_{\text{obst}}, \mu^\gamma_{\text{obst}}, \sigma^\gamma_{\text{obst}}) \leq \sum_{l=1}^{n_\gamma} p([\gamma^{l-1}_{\text{obst}}, \gamma^l_{\text{obst}}]) p^{\text{poly}}_{\text{coll}}(z_{\text{robot}}, \mu^z_{\text{obst}}, \Sigma^z_{\text{obst}}, [\gamma^{l-1}_{\text{obst}}, \gamma^l_{\text{obst}}]) \tag{3.12}$$

The sum in Equation 3.12 is easier to compute than the integral in Equation 3.11, but it is still inefficient because the obstacle orientation distribution, $\gamma_{\text{obst}} \sim \mathcal{N}(\mu^\gamma_{\text{obst}}, (\sigma^\gamma_{\text{obst}})^2)$, has infinite support. For most lane driving scenarios, the obstacle's orientation uncertainty is small relative to its total range of possible orientations and the bulk of the probability mass for γ_{obst} is grouped closely around the mean, μ^γ_{obst}, such that $\sigma^\gamma_{\text{obst}} \ll 2\pi$. Thus, the upper bound in Equation 3.12 can be reformulated as a sum over discrete angle ranges that encompass a predefined confidence interval, δ_γ, of the γ_{obst} distribution, centered around the mean. In this case $\sum_{l=1}^{n_\gamma} p([\gamma^{l-1}_{\text{obst}}, \gamma^l_{\text{obst}}]) = \delta_\gamma$. The remaining probability mass at the tails of the distribution is bounded using a circular over-

approximation.

$$
\begin{aligned}
&p_{\text{coll}}(z_{\text{robot}}, \mu^z_{\text{obst}}, \Sigma^z_{\text{obst}}, \mu^\gamma_{\text{obst}}, \sigma^\gamma_{\text{obst}}) \leq \\
&\sum_{l=1}^{n_\gamma} p([\gamma^{l-1}_{\text{obst}}, \gamma^l_{\text{obst}}]) p^{\text{poly}}_{\text{coll}}(z_{\text{robot}}, \mu^z_{\text{obst}}, \Sigma^z_{\text{obst}}, [\gamma^{l-1}_{\text{obst}}, \gamma^l_{\text{obst}}]) \\
&+ (1 - \delta_\gamma) p^{\text{circ}}_{\text{coll}}(z_{\text{robot}}, \mu^z_{\text{obst}}, \Sigma^z_{\text{obst}})
\end{aligned}
\tag{3.13}
$$

The circular bound, $p^{\text{circ}}_{\text{coll}}(z_{\text{robot}}, \mu^z_{\text{obst}}, \Sigma^z_{\text{obst}})$, is based on the bound presented by Hwang and Seah [37] and uses the furthest point on the body from the center of the back axle as the radius for both the robot and obstacle vehicles.

The right hand side of Equation 3.13 requires the computation of the probability of collision between the robot and the obstacle for a specified range of possible obstacle orientations, $p^{\text{poly}}_{\text{coll}}(z_{\text{robot}}, \mu^z_{\text{obst}}, \Sigma^z_{\text{obst}}, [\gamma^{l-1}_{\text{obst}}, \gamma^l_{\text{obst}}])$. The true instantaneous probability of collision between the robot and a predicted obstacle over a fixed range of possible orientations is calculated by integrating the obstacle position distribution, $\mathcal{N}(\mu^z_{\text{obst}}, \Sigma^z_{\text{obst}})$, over the combined body, CB, of the robot shape and all possible obstacle shapes.

$$
\begin{aligned}
p^{\text{poly}}_{\text{coll}}(z_{\text{robot}}, \mu^z_{\text{obst}}, \Sigma^z_{\text{obst}}, [\gamma^{l-1}_{\text{obst}}, \gamma^l_{\text{obst}}]) = \\
\int_{\text{CB}} \mathcal{N}(\mu^z_{\text{obst}}, \Sigma^z_{\text{obst}})
\end{aligned}
\tag{3.14}
$$

To compute an upper bound to this term, the obstacle is first rotated through its range of possible orientations and a bounding box is used to bound the obstacle shape over the entire rotation. The problem can then be converted to a single point collision calculation by sweeping the bounded obstacle shape around the perimeter of the robot rectangle to create a combined body polygon, as shown

in Figure 3.7. This is equivalent to finding the convex hull of a Minkowski sum of the robot shape and the bounded obstacle shape.

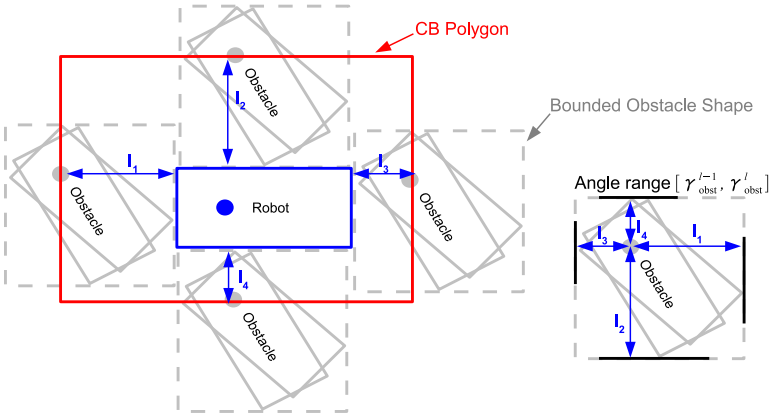


Figure 3.7: Formation of a combined body for collision probability between a rectangular robot and a rectangular obstacle with a set range of possible orientations, $\gamma_{\text{obst}} \in [\gamma^{l-1}_{\text{obst}}, \gamma^{l}_{\text{obst}}]$.

The collision probability can then be computed efficiently by exploiting the separability of Gaussian distributions to decompose the obstacle position distribution into the product of 1D Gaussian distributions. The obstacle position covariance ellipse must be aligned with the coordinate axes in order to be decoupled. However, the CB polygon must also be aligned with the coordinate axes in order to define a tight rectangular integration region. Paielli et al. [65] provide a method for normalizing the error covariance using a linear coordinate transformation:

$$W = (\sqrt{\Sigma^{z}_{\text{obst}}})^{-1} \tag{3.15}$$

where $W\Sigma^{z}_{\text{obst}}W^{T} = I_{2\times 2}$. This transformation allows the obstacle position distribution to be decoupled in any direction. An oriented bounding box is then applied to the transformed combined body polygon, CB^{W}, to define a tight rectangular integration region, and the coordinate frame is then

rotated so that the transformed combined body is aligned with the coordinate axes. This transformed, bounded and aligned CB region is defined as CB^{RW}. Decoupling the obstacle covariance along the coordinate axis allows the $p_{\mathrm{coll}}^{\mathrm{poly}}(z_{\mathrm{robot}}, \mu_{\mathrm{obst}}^z, \Sigma_{\mathrm{obst}}^z, [\gamma_{\mathrm{obst}}^{l-1}, \gamma_{\mathrm{obst}}^l])$ term to be bounded as:

$$p_{\mathrm{coll}}^{\mathrm{poly}}(z_{\mathrm{robot}}, \mu_{\mathrm{obst}}^z, \Sigma_{\mathrm{obst}}^z, [\gamma_{\mathrm{obst}}^{l-1}, \gamma_{\mathrm{obst}}^l]) \leq \int_{\mathrm{CB}^W} \mathcal{N}([\mu_{\mathrm{obst}}^x]^{RW}, 1) \cdot \mathcal{N}([\mu_{\mathrm{obst}}^y]^{RW}, 1) \tag{3.16}$$

where $[\mu_{\mathrm{obst}}^z]^{RW} = ([\mu_{\mathrm{obst}}^x]^{RW}, [\mu_{\mathrm{obst}}^y]^{RW})$ are the coordinates of the obstacle origin after the transformation and alignment.

Once aligned with the coordinate axes, the CB^{RW} bounding box is defined by the parameters $(x_{\mathrm{min}}^{\mathrm{CB}}, y_{\mathrm{min}}^{\mathrm{CB}}, x_{\mathrm{max}}^{\mathrm{CB}}, y_{\mathrm{max}}^{\mathrm{CB}})$. An upper bound on the collision probability for the given orientation range, $\gamma_{\mathrm{obst}} \in [\gamma_{\mathrm{obst}}^{l-1}, \gamma_{\mathrm{obst}}^l]$, can be computed as:

$$\begin{aligned} p_{\mathrm{coll}}^{\mathrm{rect}}(z_{\mathrm{robot}}, \mu_{\mathrm{obst}}^z, \Sigma_{\mathrm{obst}}^z, [\gamma_{\mathrm{obst}}^{l-1}, \gamma_{\mathrm{obst}}^l]) \leq \\ (\Psi(x_{\mathrm{max}}^{\mathrm{CB}} - [\mu_{\mathrm{obst}}^x]^{RW}; 0, 1) - \Psi(x_{\mathrm{min}}^{\mathrm{CB}} - [\mu_{\mathrm{obst}}^x]^{RW}; 0, 1)) \\ \cdot (\Psi(y_{\mathrm{max}}^{\mathrm{CB}} - [\mu_{\mathrm{obst}}^y]^{RW}; 0, 1) - \Psi(y_{\mathrm{min}}^{\mathrm{CB}} - [\mu_{\mathrm{obst}}^y]^{RW}; 0, 1)) \end{aligned} \tag{3.17}$$

where $\Psi(z; \mu, \sigma) = \frac{1}{2}[1 + \mathrm{erf}(\frac{z-\mu}{\sqrt{2\sigma^2}})]$ is the 1D Gaussian cumulative distribution function.

The steps of the polygonal collision probability bound calculation for an arbitrary polygonal robot shape and a rectangular obstacle are shown in Figure 3.8. The top left image shows the initial scene, with a bounded obstacle shape over the given orientation range, $[\gamma_{\mathrm{obst}}^{l-1}, \gamma_{\mathrm{obst}}^l]$. The top right image shows the combined body of the robot shape and the bounded obstacle shape, constructed as shown in Figure 3.7. The bottom left image shows the effects of the linear transformation W, and the bottom right image shows the final configura-

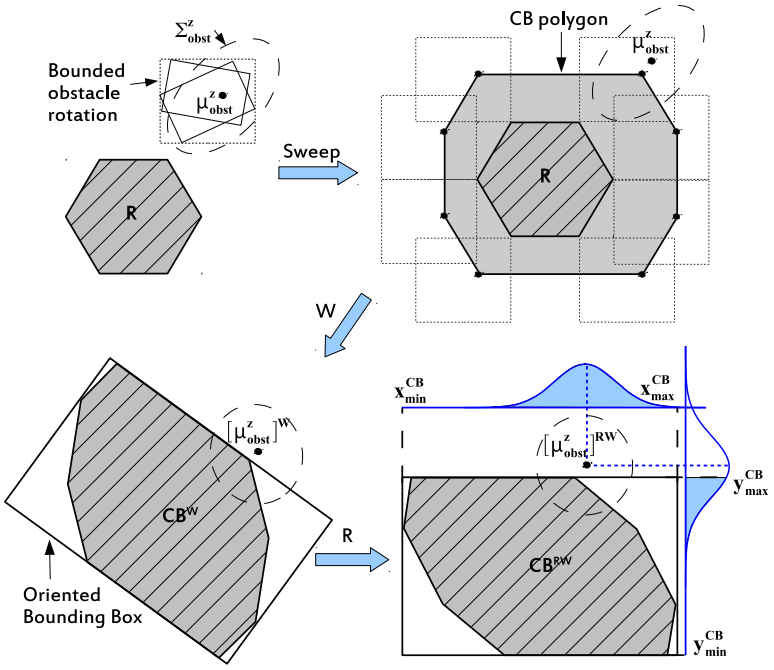


Figure 3.8: Depiction of polygonal collision probability bound calculation between an arbitrary polygon region and a rectangular obstacle for a known range of obstacle orientations. Top Left: the initial scene. Top Right: the combined body of the obstacle shape and the vehicle shape. Bottom Left: the effects of the transformation W and the application of an oriented bounding box. Bottom Right: integration over the aligned combined body, CB^{RW}, using the decoupled, normalized obstacle position distribution.

tion after a rotation is applied to align the transformed CB with the coordinate axes of the reference frame.

Collision Probability Analysis

The polygonal collision probability bound, $p_{\mathrm{coll}}^{\mathrm{poly}}(z_{\mathrm{robot}}, \mu_{\mathrm{obst}}^{z}, \Sigma_{\mathrm{obst}}^{z}, \mu_{\mathrm{obst}}^{\gamma}, \sigma_{\mathrm{obst}}^{\gamma})$ in Equation 3.13, provides a tighter bound than a circular collision probability bound, but this improvement in accuracy comes at a higher computational cost.

In order to understand this trade-off, a close proximity obstacle interaction scenario is evaluated. Figure 3.9 shows an interaction, in a robot centric coordinate frame, between the robot vehicle and an obstacle trajectory prediction. A comparison is made for each point on the trajectory, $k \in [1, 10]$, for three cases: a circular bound, and two versions of the polygonal bound (Eqn. 3.13 using $n_\gamma = 1$ and $n_\gamma = 5$). The true probability of collision is also computed using Monte Carlo simulations. To improve computational performance, Equation 3.13 is simplified by setting $p_{\text{coll}}^{\text{circ}}(z_{\text{robot}}, \mu_{\text{obst}}^z, \Sigma_{\text{obst}}^z) = 1$ and $\delta_\gamma = 0.99$. This simplification eliminates the need to compute $p_{\text{coll}}^{\text{circ}}(z_{\text{robot}}, \mu_{\text{obst}}^z, \Sigma_{\text{obst}}^z)$ and is accurate for δ_γ values close to one. This simplification is also used in all subsequent simulations.

Figure 3.9 shows that the polygonal collision probability bound produces a much tighter upper bound than the circular collision probability bound, especially at $k = 4$, where the obstacle vehicle is driving parallel to the robot in an adjacent lane. This tighter upper bound still prevents unsafe situations, yet also allows expected close proximity driving behaviors. The results in Figure 3.9 also show that $n_\gamma = 1$ provides nearly as tight of a bound as $n_\gamma = 5$, but with a computational cost nearly equivalent to that of the circular bound. The polygonal bound performs the worst at $k = 6$, when the obstacle is aligned at a nearly 45° angle in the robot's reference frame. This misalignment causes the bounding box approximations to poorly fit the obstacle shape. However, cases such as $k = 6$ only occur when the obstacle is blocking the robot's lane of travel, and in these situations, substantial over-approximations are acceptable.

To evaluate how the polygonal collision probability bound varies as a function of the obstacle angle uncertainty, the $k = 4$ case was reevaluated for

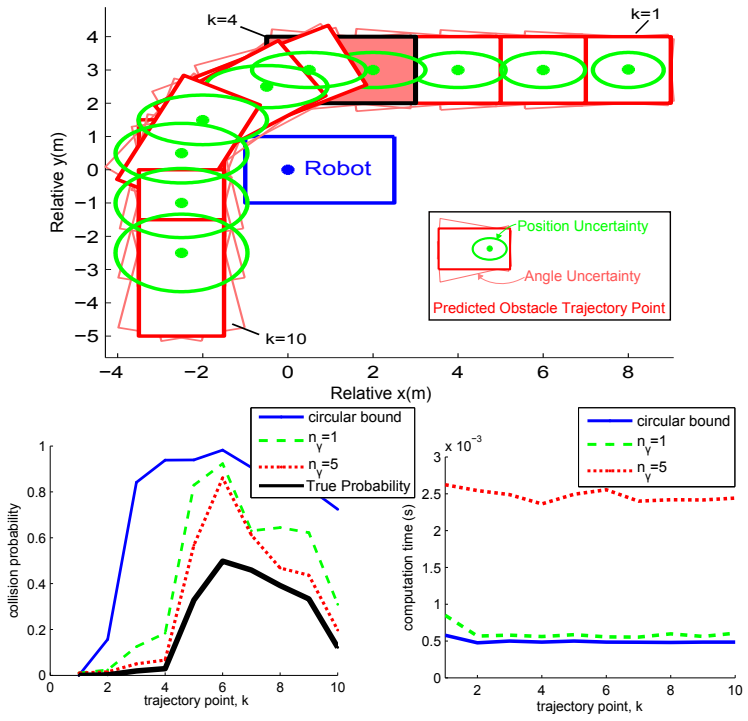


Figure 3.9: Comparison of different collision probability approximations for a sample close proximity obstacle interaction scenario. Top: the relative trajectory of the robot and obstacle vehicles. Bottom Left: collision probability comparison for each trajectory point. Bottom Right: computation time comparison.

$n_\gamma \in [1, 10]$ over a range of obstacle angle standard deviations, $\sigma^\gamma_{\text{obst}}$, as shown in Figure 3.10. The results show that when $n_\gamma > 3$, the polygonal collision probability bound is tight compared to the true collision probability over the entire tested range of obstacle angle standard deviations, $\sigma^\gamma_{\text{obst}} \in [3°, 43°]$. The $n_\gamma = 1$ bound is most useful for cases with low obstacle angle uncertainty, $\sigma^\gamma_{\text{obst}} < 20°$; however, the $n_\gamma = 1$ bound still offers a significant improvement over the circular bound for $\sigma^\gamma_{\text{obst}} < 35°$. The polygonal bound does not, in general, converge to the true probability as $n_\gamma \to \infty$; over-approximations still occur due to successive bounding box approximations. A constant offset is also introduced by using $p^{\text{circ}}_{\text{coll}}(z_{\text{robot}}, \mu^z_{\text{obst}}, \Sigma^z_{\text{obst}}) = 1$ in Equation 3.13.

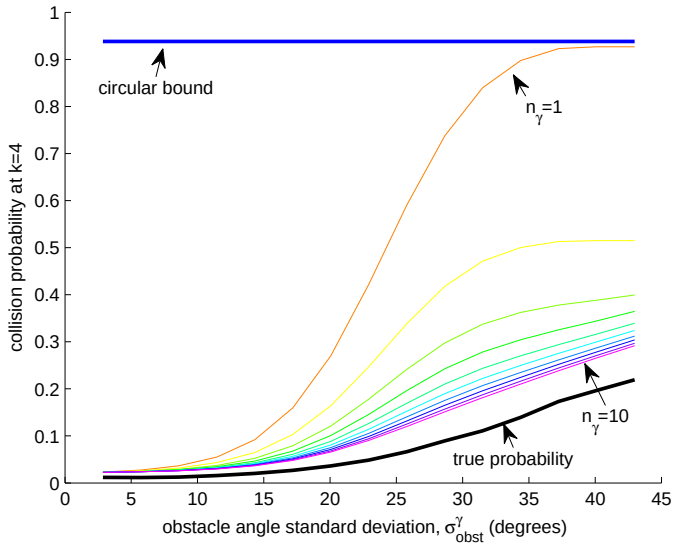


Figure 3.10: Collision probability bound for $k = 4$ as a function of standard deviation of the obstacle angle uncertainty, $\sigma^{\gamma}_{\text{obst}}$, and number of orientation ranges, n_{γ}.

3.3.5 Optimization Summary

The contingency planner uses a nonlinear, constrained numerical optimization procedure, as defined in Equation 3.5, to find a locally optimal set of contingency paths, parameterized by $\mathbf{h}$. The appealing aspect of this constrained optimization approach is the formulation: the optimization cost function can be defined based on desirable path goals and behaviors, and optimization constraints can be defined to enforce safety requirements and to ensure conformance with vehicle limitations.

These costs are defined as a sum over discrete evaluation points, $k \in [1, N]$. For this paper, the time interval between evaluation points, Δt_{eval}, is set equal to the obstacle prediction time step for convenience. This discrete formulation approximates a continuous integral over the path segments, but allows for simpler

point-wise calculations.

The cost for the shared segment, $J_{\text{shared}}(\mathbf{h}, \mathbf{T})$, is defined as:

$$
\begin{aligned}
&J_{\text{shared}}\left(\mathbf{h}, \mathbf{T}\right) = \\
&\underbrace{\alpha_a \sum_{k=1}^{n_1}((a_k^x)^2 + (a_k^y)^2) + \alpha_c \sum_{k=1}^{n_1}(v_k^x a_k^y - v_k^y a_k^x)^2 +}_{J_{\text{shared}}^{\text{dynamics}}} \\
&\underbrace{\alpha_d \sum_{k=1}^{n_1} f_{\text{prx}}(x_k, y_k) +}_{J_{\text{shared}}^{\text{static}}} \\
&\underbrace{\alpha_p \sum_{\mathcal{T}^j \in \mathbf{T}} p(\mathcal{T}^j) \sum_{T \in \mathcal{T}^j} \sum_{k=1}^{n_1} f_{\text{coll}}^{\text{circ}}(x_k, y_k, T(k))}_{J_{\text{shared}}^{\text{collision}}}
\end{aligned}
\tag{3.18}
$$

and the cost for each contingency path, $J_{\text{conting}}(\mathbf{h}, \mathcal{T}^j)$ is defined as:

$$
\begin{aligned}
&J_{\text{conting}}\left(\mathbf{h}, \mathcal{T}^j\right) = \\
&\underbrace{\alpha_a \sum_{k=n_1+1}^{N}((a_k^x)^2 + (a_k^y)^2) + \alpha_c \sum_{k=n_1+1}^{N}(v_k^x a_k^y - v_k^y a_k^x)^2 +}_{J_{\text{conting}}^{\text{dynamics}}} \\
&\underbrace{\alpha_d \sum_{k=n_1+1}^{N} f_{\text{prx}}(x_k, y_k)}_{J_{\text{conting}}^{\text{static}}} \underbrace{+ \alpha_p \sum_{T \in \mathcal{T}^j} \sum_{k=n_1+1}^{N} f_{\text{coll}}^{\text{circ}}(x_k, y_k, T(k))}_{J_{\text{conting}}^{\text{collision}}} \\
&\underbrace{+ \alpha_g f_{\text{goal}}(x_N, y_N)}_{J_{\text{conting}}^{\text{goal}}}
\end{aligned}
\tag{3.19}
$$

where n_1 is the number of discrete timesteps evaluated over the initial path segment and $N - n_1$ represents the number of discrete timesteps evaluated along each contingency path. The f_{prx} term penalizes proximity to static obstacles as defined in Equation 3.10, f_{goal} is the squared Euclidean distance between a given path point and the goal point, and the $f_{\text{coll}}^{\text{circ}}$ term is a circular collision probability bound based on the bound in [37]. This circular bound is equivalent to the

collision probability bound presented in Section 3.3.4, where both the robot and the obstacle shapes are approximated by circles. Here, $f_{\text{coll}}^{\text{circ}}$ is used as an under-approximation of the true collision probability, where the width of the robot and obstacle are used as their radii. This under-approximation results in low penalties on expected close proximity obstacle interactions while allowing for an easier to optimize, orientation independent cost term. This under-approximation is permissible because true collision probability bounds are used in the optimization constraints to ensure path safety.

The $\alpha_{(\cdot)}$ parameters in Equations 3.18-3.19 are weights that govern the relative importance of each penalty term: the α_a weighted term penalizes high acceleration, the α_c weighted term penalizes unnormalized path curvature, the α_d weighted term penalizes proximity to static obstacles, the α_p weighted term penalizes high collision probabilities with dynamic obstacles, and the α_g weighted term penalizes the distance of the robot from its goal at the end of the planning horizon.

The acceleration, curvature, and distance from goal terms are formulated to be convex with respect to position, (x, y), and its derivatives, (v^x, v^y, a^x, a^y). Unnormalized path curvature, $(v_k^x a_k^y - v_k^y a_k^x)$, is used to maintain convexity and is equivalent to weighting the path curvature by the robot velocity magnitude cubed. The collision probability term $f_{\text{coll}}^{\text{circ}}$ represents a unimodal cost hill in the robot's (x, y) space and behaves as a convex function away from the cost peak. The static obstacle proximity term, $f_{\text{prx}}(x_k, y_k)$, is convex with respect to the robot's lateral offset from the center of the driving corridor.

A set of constraint functions, $C(\mathbf{h})$, are evaluated at each timestep k along the shared initial segment and along each contingency path, in order to ensure the

optimized solution obeys the physical constraints of the robot and stays below a required maximum probability of collision with dynamic obstacles. These constraint functions are:

$$-(v_{k-1}^x v_k^x - v_{k-1}^y v_k^y) - \beta_{\max}||\mathbf{v}_{k-1}|| \cdot ||\mathbf{v}_k|| \leq 0 \tag{3.20}$$

$$(v_k^x)^2 + (v_k^y)^2 - v_{\max}^2 \leq 0 \tag{3.21}$$

$$\left(\frac{(a_k^x)^2 + (a_k^y)^2}{[a_f]_{\max}^2}\right) + \left(\frac{(v_k^x a_k^y - v_k^y a_k^x)^2}{\epsilon^2 [a_l]_{\max}^2}\right) - 1 \leq 0 \tag{3.22}$$

$$f_{\text{prx}}(x_k, y_k) - 1 \leq 0 \tag{3.23}$$

$$-p_{\max} + \sum_{T \in \mathcal{T}^j} p(T) f_{\text{coll}}^{\text{poly}}(x_k, y_k, \theta_{\text{robot},k}, T(k)) \leq 0 \tag{3.24}$$

Equation 3.20 constrains changes in robot orientation between successive timesteps, where the parameter $\beta_{\max}$ controls the maximum allowable change in orientation (e.g. $\beta_{\max} = 0$ equates to $[\Delta\theta_{\text{robot}}]_{\max} = 90°$). Equation 3.21 enforces a maximum velocity constraint. Equation 3.22 enforces a maximum acceleration constraint using an acceleration ellipse, where $[a_f]_{\max}$ is the maximum allowable forward acceleration, $[a_l]_{\max}$ is the maximum allowable lateral acceleration, and $(v_k^x a_k^y - v_k^y a_k^x)$ is an approximation of the robot's lateral acceleration, which is equivalent to the robot's lateral acceleration multiplied by the velocity magnitude. Since $(v_k^x a_k^y - v_k^y a_k^x)$ is an approximation, the constant $\epsilon = 1m/s$ is introduced to ensure consistent units. Equation 3.23 constrains the robot's configuration space to prevent collisions with static obstacle regions. Equation 3.24 constrains the maximum probability of collision between the robot following the j^{th} contingency plan and the dynamic obstacle trajectories in $\mathcal{T}^j$ to be less than $p_{\max}$ using the collision probability bound defined in equation 3.13, where $z_{\text{robot}} = [x_k, y_k]$ and $\theta_{\text{robot},k}$ are used to transform the calculation into a robot

centric coordinate frame, and $\mu^z_{\text{obst}}, \Sigma^z_{\text{obst}}, \mu^\gamma_{\text{obst}}, \sigma^\gamma_{\text{obst}}$ are defined by $T(k)$.

The $f^{\text{poly}}_{\text{coll}}$ constraint function in Equation 3.24 includes orientation information, $\theta_{\text{robot},k} = \tan^{-1}(\frac{v^y_k}{v^x_k})$, making this term non-convex. However, this constraint is necessary because it provides a formal bound on collision probability and ensures the safety of the final set of contingency paths. This term ensures all feasible optimization solutions are safe up to a probability of $p_{\max}$. A circular collision probability bound could be used here to improve convexity; however, as described earlier, the circular probability bound is overly conservative and prevents normal driving behaviors such as passing in adjacent road lanes.

3.4 Trajectory Clustering for Improved Computational Scaling

Equation 3.6 and Equation 3.9 show respectively that both the cost function complexity and the number of optimization variables required by the proposed contingency planning algorithm scale linearly with the number of obstacle prediction permutations, n_s. In turn, Equation 3.3 shows that n_s scales exponentially with the number of obstacles, n_o, and the number of trajectory predictions for each obstacle, $|\mathcal{T}_i|$. This exponential scaling effectively prevents any reasonably complex contingency planning algorithm, with existing levels of computational power, from performing real-time navigation in environments with more than a small number of obstacles and possible goal states.

The number of required contingency plans can be reduced by clustering obstacle prediction permutations prior to executing the Contingency Planner based on similarities in their potential influence on the robot's future path. This allows the planner to use a fixed maximum number of contingency plans, n_c, re-

gardless of the number of obstacles and possible obstacle goals in the environment. Clustering similar predicted obstacle trajectory permutations provides improved computational scaling because it allows the planner to ignore combinatorial permutations of obstacle predictions within trajectory clusters, under the assumption that these permutations all have a similar influence on the robot's future path and can be safely approximated as simultaneously occurring events. The planner is thus able to treat each cluster of trajectory permutations, $\mathcal{C}^l$, as the union of all possible trajectories contained in its constituent permutations, $\mathcal{C}^l = \bigcup_{\mathcal{T}^j \in \mathbf{T}^l} \mathcal{T}^j$, where $\mathbf{T}^l \subset \mathbf{T}$ represents the set of obstacle trajectory permutations included in the l^{th} cluster.

Trajectory clustering reduces the complexity of the planning problem from planning an exponential n_s number of contingency plans that each need to avoid n_o obstacle trajectories, to planning a fixed n_c number of contingency plans, that each must avoid a linearly scaling $\left|C^l\right| = \sum_{i=1}^{n_o} \left|\mathcal{T}_i^l\right|$ number of obstacle trajectories, where $\mathcal{T}_i^l \subset \mathcal{T}_i$ is the subset of possible trajectory predictions for obstacle i included in the l^{th} cluster. In the limit of clustering all predicted obstacle trajectories into a single cluster, $n_c = 1$, the Motion Planner would recover the single path dynamic planning approach depicted in Figure 3.2(a), where $|\mathcal{C}^1| = \sum_{i=1}^{n_o} |\mathcal{T}_i|$.

3.4.1 Trajectory Clustering

Clustering spatial trajectories is a popular a machine learning technique for modeling the motion of people or other obstacles through a given environment. Hierarchical methods are commonly used with trajectory clustering since sim-

ple generative models are difficult to formulate. Fu, Hu, and Tan [22] present a clustering method using both spectral and hierarchical clustering for identifying common trajectories in traffic videos. Lee, Han, and Whang [54] identify similar sub-trajectories in a group of trajectories by first partitioning each trajectory into a sequence of line segments and then clustering the line segments using a hierarchical joining method. Non-hierarchical approaches have also been developed, including Gaffney and Smyth [23] who assume a generative model for a set of trajectories and use the EM algorithm to learn the mixture assignments. All of these approaches effectively group similar trajectories, but they have a fundamentally different goal than a clustering algorithm designed to simplify path planning.

The clustering algorithm outlined in this section is designed to maximize dissimilarity between clustered sets of obstacle trajectory predictions in order to capture the most important mutually exclusive obstacle decisions and uncertainties with a smaller set of contingency plans, allowing the robot to maintain the performance advantages of exhaustive contingency planning while requiring less computation time. Figure 3.11 shows the Contingency Planning architecture with Trajectory Clustering included as a preprocessing step between Obstacle Prediction and Contingency Planning.

Using a fixed number of clusters, n_c, where $n_c \leq n_s$, Equation 3.6 can be rewritten as:

$$J(\mathbf{h}) = J_{\text{shared}}(\mathbf{h}, \mathbf{C}) + \sum_{\mathcal{C}^l \in \mathbf{C}} p(\mathcal{C}^l) J_{\text{conting}}(\mathbf{h}, \mathcal{C}^l) \tag{3.25}$$

where $\mathbf{C} = \{\mathcal{C}^1, \ldots, \mathcal{C}^{n_c}\}$ is a possible clustering of the obstacle trajectory predictions.

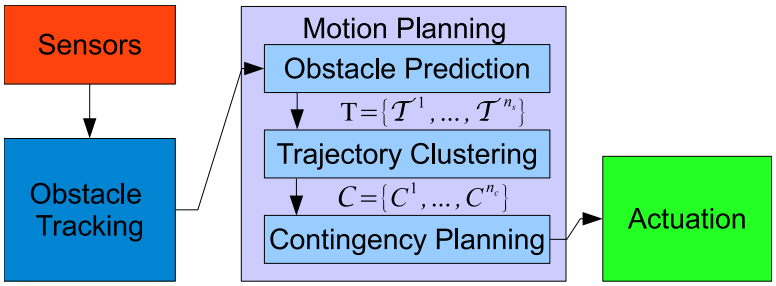


Figure 3.11: Block diagram of the contingency planning architecture with trajectory clustering.

Similarly, the dynamic obstacle cost term in J_{shared} can be redefined as:

$$J_{\text{shared}}^{\text{collision}} = \alpha_p \sum_{\mathcal{C}^l \in \mathbf{C}} p(\mathcal{C}^l) \sum_{T \in \mathcal{C}^l} \sum_{k=1}^{n_1} f_{\text{coll}}^{\text{circ}}(x_k, y_k, T(k)) \tag{3.26}$$

and the dynamic obstacle cost term in J_{conting} can be redefined as:

$$J_{\text{conting}}^{\text{collision}} = \alpha_p \sum_{T \in \mathcal{C}^l} \sum_{k=n_1+1}^{N} f_{\text{coll}}^{\text{circ}}(x_k, y_k, T(k)) \tag{3.27}$$

The dynamic collision probability optimization constraint in Equation 3.24 can also be redefined as:

$$-p_{\max} + \sum_{T \in \mathcal{C}^l} p(T) f_{\text{coll}}^{\text{poly}}(x_k, y_k, \theta_{\text{robot},k}, T(k)) \leq 0 \tag{3.28}$$

The goal of the trajectory clustering algorithm is to maximize dissimilarity between the obstacle predictions in different clusters. A probability weighted, minimum relevant dissimilarity metric is formulated to capture the dissimilarity between predicted obstacle trajectories, along with the relative occurrence probabilities of the obstacle predictions in each cluster, and the potential impact or relevance each obstacle prediction has on the robot's potential path. For clusters $\mathcal{C}^a, \mathcal{C}^b$, where $a, b \in [1, n_c]$, this metric is defined as:

$$\text{diss}(\mathcal{C}^a, \mathcal{C}^b) = p(\mathcal{C}^a, \mathcal{C}^b) \cdot \min_{T^a \in \mathcal{C}^a} \left(w^a \cdot \min_{T^b \in \mathcal{C}^b} D(T^a, T^b) \right) \tag{3.29}$$

where $p(\mathcal{C}^a, \mathcal{C}^b)$ is the joint probability of the obstacle predictions in both clusters, w^a is a relevancy weighting based on how likely predicted obstacle trajectory T^a is to affect the robot's nominal path:

$$w^a = \frac{p(T^a)}{\min_k (\mu^a(k) - z_{\text{robot}}(k))^T \Sigma^a(k)^{-1} (\mu^a(k) - z_{\text{robot}}(k))} \tag{3.30}$$

and $D(T^a, T^b)$ is a dissimilarity metric that reflects the dissimilarity between two predicted obstacle trajectories.

Using this dissimilarity definition, the optimal set of obstacle prediction trajectory clusters is defined as:

$$\mathbf{C}_{\text{opt}} = \arg\max_{\mathbf{C}} \left(\min_{\mathcal{C}^a, \mathcal{C}^b \in \mathbf{C}} \text{diss}(\mathcal{C}^a, \mathcal{C}^b) \right) \tag{3.31}$$

Dissimilarity Metric

The dissimilarity metric, $D(T^a, T^b)$, in Equation 3.29 represents the likelihood of two predicted obstacle trajectories having a similar influence on the robot's free configuration space. The potential influence of a predicted obstacle trajectory on the robot's free configuration space is found in an efficient manner by representing the robot's driving corridor, $\mathbf{D}_{\text{corr}}$, as the interior points of a set of convex polygons. The set of driving corridor polygons is derived by decomposing the driving corridor definition presented in Section 3.3.3 into a set of trapezoids, $\mathcal{R}_1$ through $\mathcal{R}_{n_R}$. The polygonal collision probability bound presented in Equation 3.13 is then applied to each polygon $\mathcal{R}_{(\cdot)}$ in $\mathbf{D}_{\text{corr}}$ to provide a fast probability bound on whether each discrete timestep along a predicted obstacle trajectory, $T(k)$, lies within the driving corridor. This probability bound is then thresholded to determine whether $T(k) \in \mathbf{D}_{\text{corr}}$.

Using this driving corridor inclusion definition, the dissimilarity metric, $D(T^a, T^b)$, can be defined as:

$$D(T^a, T^b) = \frac{1}{N} \sum_{k=1}^{N} d(T^a(k), T^b(k)) \tag{3.32}$$

where N is the total number of points in each trajectory and $d(T^a(k), T^b(k))$ is the dissimilarity between two 2D obstacle position distributions, $T^{(\cdot)}(k) \sim \mathcal{N}(\mu^{(\cdot)}, \Sigma^{(\cdot)})$, defined as:

$$d(T^a(k), T^b(k)) = \begin{cases} 1 - \frac{1}{e^{M(T^a(k), T^b(k))}} & : T^a(k) \wedge T^b(k) \in \mathbf{D}_{\text{corr}} \\ 1 & : T^a(k) \oplus T^b(k) \in \mathbf{D}_{\text{corr}} \\ 0 & : T^a(k) \wedge T^b(k) \notin \mathbf{D}_{\text{corr}} \end{cases} \tag{3.33}$$

where $M(T^a(k), T^b(k))$ is the squared Mahalanobis distance between $T^a(k)$ and $T^b(k)$:

$$M(T^a(k), T^b(k)) = (\mu^a - \mu^b)^T (\Sigma^a + \Sigma^b)^{-1} (\mu^a - \mu^b) \tag{3.34}$$

Trajectory points which coincide within the driving corridor have zero dissimilarity. As two trajectory points diverge within the driving corridor their dissimilarity grows until it reaches a maximum at $d(T^a(k), T^b(k)) = 1$ for two points that have no chance of overlapping. A trajectory point within the driving corridor has a dissimilarity measure of $d(T^a(k), T^b(k)) = 1$ when compared with any point outside of the driving corridor, and two trajectory points that lie

entirely outside of the robot's driving corridor have zero dissimilarity, meaning that the clustering algorithm has no incentive to separate them.

Hierarchical Splitting Algorithm

The clustering optimization problem given in Equation 3.31 can be solved efficiently by using a hierarchical splitting approach that exploits the underlying structure of the problem. Since the clustering algorithm is performed over the elements of $\mathbf{T}$, which is a Cartesian product of the predicted obstacle trajectories for each obstacle, it is only possible to isolate a given obstacle trajectory from the other possible trajectories for the same obstacle. This is equivalent to splitting along a single dimension of the Cartesian product. This implies that, for any cluster of trajectory permutations, $\mathcal{C}^l$, a selected subset of obstacle trajectories for obstacle i, $\tau^* \subset \mathcal{T}_i^l$, can be isolated by splitting the total cluster into new clusters, where one cluster contains τ^* and the other contains $\mathcal{T}_i^l \setminus \tau^*$. Both clusters also inherit all of the trajectories from the other obstacles.

Clustering is thus performed by initializing a single cluster and performing successive greedy splits in order to iteratively isolate the subset of trajectories from a single obstacle, τ^*, with the largest probability weighted, relevant dissimilarity when compared to the rest of the trajectories in the cluster. This approach makes the best greedy split possible, based on the probability weighted, relevant dissimilarity metric defined in Equation 3.29, while still preserving all possible obstacle trajectory permutations.

3.5 Dangerous Intersection Simulations

Monte Carlo simulations were conducted using a dangerous intersection scenario in order to study the safety and performance of the proposed contingency planning algorithm on a robot engaging in complex obstacle avoidance interactions. This dangerous intersection scenario consists of a robot and one or more obstacle vehicles navigating a four-way intersection with no stop signs or other traffic rules. These simulations represent a worst case intersection scenario and are designed to ensure efficient use of simulation runs by consistently forcing the robot into interesting obstacle avoidance interactions. Due to the condensed number of risky encounters, more aggressive braking and higher collisions frequencies are expected compared to normal driving. Collision frequency results are presented here as a relative metric of safety performance between path planning algorithms. Sections 3.5.1 and 3.5.2 present simulation results for a single obstacle vehicle, while Section 3.5.3 presents two sets of simulation results for multiple obstacle vehicles; a congested multiple obstacle case is presented in Section 3.5.3 and a less-congested multiple obstacle case is presented in Section 3.5.3.

In all simulations, obstacle vehicles are assigned a non-intelligent driving behavior. This means that they completely ignore the robot while navigating the intersection, even if a collision is imminent. This obstacle behavior eliminates the effects of complex reciprocal robot/obstacle anticipation interactions and allows for consistent analysis of the path planner performance. Obstacle vehicles use a simplified, constraint free version of the numerical optimization based path planner presented in Section 3.3 (Equations 3.18-3.19), but with the $J_{(\cdot)}^{\text{collision}}$ collision probability terms omitted. This same simplified path planner

is used as the obstacle prediction model, and each potential obstacle goal, (Turn Left, Turn Right, Go Straight), is initialized with equal likelihood.

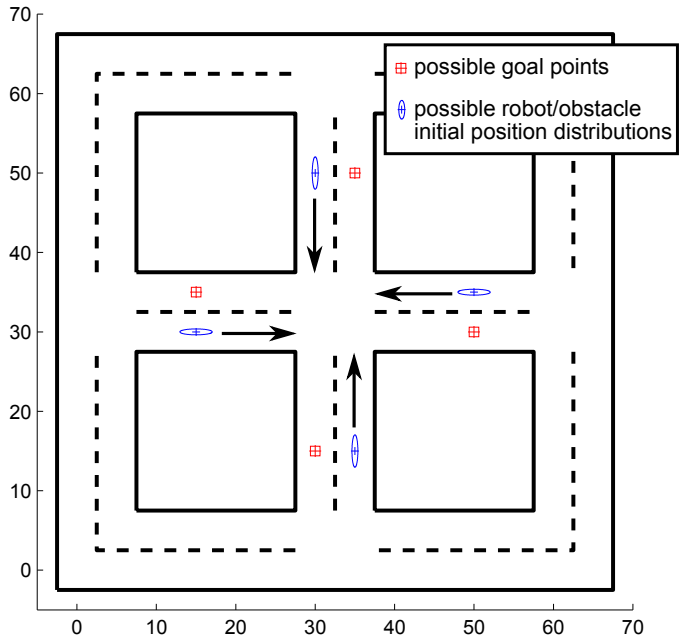


Figure 3.12: Map of simulated scenario with possible initial robot/obstacle distributions depicted by their covariance ellipses and possible goal points depicted as squares.

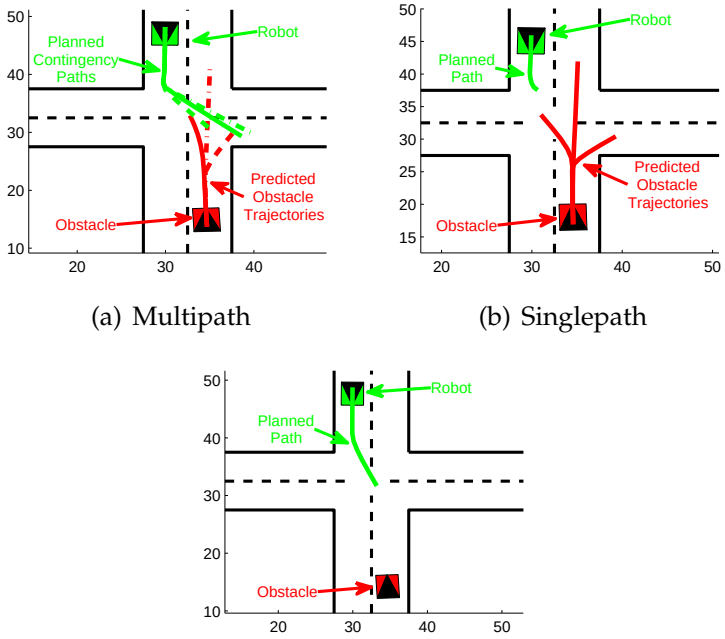
3.5.1 Single Obstacle Simulation Results

The initial conditions for the single obstacle simulations are depicted in Figure 3.12. The states of both the robot and the obstacle vehicle are randomly initialized along one of the four road segments leading into the intersection and both are randomly assigned one of four goal points along the road segments leaving the intersection. Goal point selection is constrained to avoid U-turn maneuvers. The possible initial position distributions and goal points are depicted

in Figure 3.12. The obstacle vehicle is assigned a mean initial velocity of $5\mathrm{m/s}$ and the robot is assigned a mean initial velocity of $3\mathrm{m/s}$. This difference in the initial velocity distributions biases the simulation towards more interesting cases where the obstacle arrives at the intersection before the robot, forcing the robot to actively react to and avoid the obstacle. Cases where the robot arrives at the intersection sufficiently before the obstacle vehicle simply result in the robot traversing the intersection unimpeded.

As a baseline comparison, three algorithms were run: 1) Multipath: the contingency approach proposed in this paper which considers all possible obstacle predictions individually (this is equivalent to using trajectory clustering with $n_c = 3$), 2) Singlepath: a single path variation of the Multipath planner where a single contingency path is used that avoids all obstacle predictions (equivalent to $n_c = 1$), and 3) Static: a variation of the Singlepath planner which uses a stationary obstacle prediction motion model (obstacles are predicted to not move). All other optimization parameters are identical. Both the obstacle vehicle and the robot use a pure pursuit based actuation controller [12] to follow their planned paths. Simulations were run in MATLAB on a 3GHz Core i3 processor running under Windows 7 and the path optimization solution was written in C++ using the open source nonlinear optimization library IPOPT [81].

Figure 3.13 shows a sample planning result for each of these algorithms. The Multipath planner, shown in Figure 3.13(a), produces an aggressive plan for traversing the intersection since its multiple contingency paths can account for each possible obstacle action independently. The Singlepath planner, shown in Figure 3.13(b), is forced to take a conservative approach to the intersection since it must avoid all possible obstacle trajectories simultaneously. The Static



(a) Multipath (b) Singlepath

(c) Static

Figure 3.13: Sample planning result for each of the three tested algorithms during a single planning cycle.

planner, shown in Figure 3.13(c), plans a clearly unsafe path because it is unable to anticipate the future motion of the obstacle.

A total of $n_{\text{sim}} = 300$ simulations were run, each 8 seconds in length, using a maximum probability of collision threshold of $p_{\max} = 10\%$. This $p_{\max}$ threshold was found to provide a good balance between driving performance and safety. Four performance metrics were recorded: collision frequency, minimum distance to obstacle, average squared acceleration, and minimum distance to goal. Statistics for these metrics, over the $n_{\text{sim}} = 300$ runs, are listed in Table 3.1, where $\mu_{(\cdot)}$ is the mean, and $SE_{(\cdot)}$ is the standard error of the mean, $SE_{(\cdot)} = \frac{\sigma_{(\cdot)}}{\sqrt{n_{\text{sim}}}}$. Table 3.1 lists both the total percentage of simulations that resulted in a collision, $n_{\text{coll}}^{\text{tot}}$, and the percentage of at-fault collisions, $n_{\text{coll}}^{\text{fault}}$. At-fault collisions occur

Table 3.1: Performance metrics for $n_{\text{sim}} = 300$ single obstacle simulation runs

			min. distance to obstacles		avg. squared acceleration		min. distance to goal	
	$n_{\text{coll}}^{\text{tot}}$	$n_{\text{coll}}^{\text{fault}}$	μ_{obst}	SE_{obst}	μ_{acc}	SE_{acc}	μ_{goal}	SE_{goal}
Multi	2.00	0.00	3.81	0.155	1.43	0.065	12.13	0.33
Single	2.00	0.33	4.43	0.190	1.25	0.058	12.80	0.35
Static	18.00	10.0	2.63	0.151	1.57	0.064	11.36	0.36
units	%	%	(m)		$(\text{m/s}^2)^2$		(m)	

when the robot is genuinely behaving dangerously and are defined as collisions where the robot is still moving at a non-negligible speed, $|v_{\text{robot}}| > 0.1\text{m/s}$, at the time of the collision. Collisions that do not meet this criteria are considered not-at-fault because they likely would not result in a collision if the obstacle vehicle had any avoidance capabilities.

The collision frequency results in Table 3.1 show that both the Multipath and Singlepath planners experience far fewer total and at-fault collisions than the Static planner. Table 3.1 also shows that the Multipath planner is more aggressive in terms of obstacle spacing and speed through the intersection than the Singlepath planner, but that it achieves this performance increase at the cost of increased acceleration effort. Table 3.2 shows the paired *t*-test results for each metric, indicating whether the difference in performance between the Multipath planner and the Static and Singlepath planners are statistically significant. Acceptance of the null hypothesis, H = 0, indicates that the difference between the two cases is not statistically significant. These tests indicate that all of performance differences seen in Table 3.1 are statistically significant at the $\alpha = 0.05$

Table 3.2: Paired t-test results for single obstacle simulations at $\alpha = 0.05$

	min. distance to obstacle		avg. squared acceleration		min. distance to goal	
comparison	H	P	H	P	H	P
Singlepath vs. Multipath	1	< 0.001	1	< 0.001	1	0.001
Static vs. Multipath	1	< 0.001	1	0.041	1	0.003

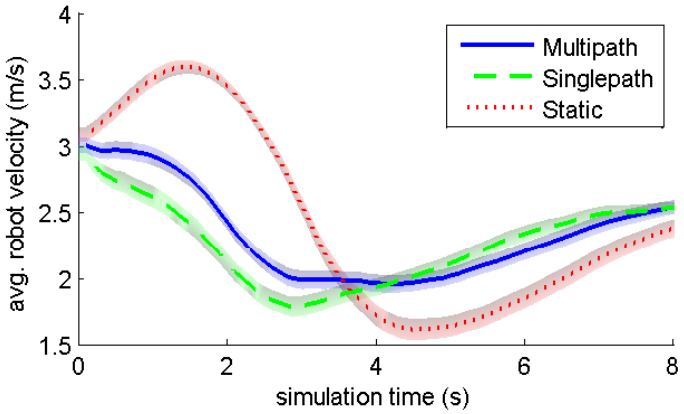


Figure 3.14: Average velocity as a function of simulation time. Shaded regions reflect the one standard error of the mean (SE_{vel}) uncertainty bounds.

confidence level.

Figure 3.14 plots the velocity history of the robot averaged over all $n_{\text{sim}} = 300$ simulations. This plot shows that the Multipath algorithm is more aggressive that the Singlepath planner at the start of the simulation, due to its less conservative assumptions about the obstacle's future motion. The Singlepath planner is less aggressive because it assumes all mutually exclusive obstacle predictions are simultaneously occurring events. As the robot approaches the intersection during a typical simulation run, the predicted obstacle trajectories cause the

intersection to appear completely blocked, forcing the robot to apply its brakes immediately. The Multipath planner, alternatively, is able to recognize that there may be a clear path by the time the robot reaches the intersection and allows the robot to postpone braking until it is necessary for safety reasons. This delayed braking is visible in the $t = 1$ to $t = 3$ second range of Figure 3.14.

These results show that, while the performance of the Singlepath planner is similar to the Multipath planner in terms of collision avoidance, the Singlepath planner exhibits a more conservative driving style. The Multipath algorithm performs much closer to the Static algorithm in terms of driving aggressiveness, but is significantly safer, experiencing far fewer close calls and collisions.

3.5.2 Sensitivity to Collision Threshold

The maximum probability of collision threshold, $p_{\max}$ in Equation 3.24, limits the robot's aggressiveness with respect to obstacle interactions. To investigate the effect of this parameter on the robot's performance, $n_{\text{sim}} = 300$ simulations were run for a range of $p_{\max}$ values from 1% to 50%. Figures 3.15-3.19 plot the average collision frequency and performance metrics as a function of $p_{\max}$ for the Multipath, Singlepath, and Static planners.

Figures 3.15-3.16 plot the total and at-fault collision frequencies of each planning algorithm as a function of $p_{\max}$. Lines corresponding to $n_{\text{coll}}^{\text{tot}} = p_{\max}$ and $n_{\text{coll}}^{\text{fault}} = p_{\max}$ are also plotted for reference. Figure 3.15 shows that at low values of $p_{\max}$ ($< 10\%$), there is little difference in total collision frequency between the Multipath and the Singlepath planners, while at high values of $p_{\max}$ ($> 40\%$), both planners begin to approach the collision frequency of the static planner.

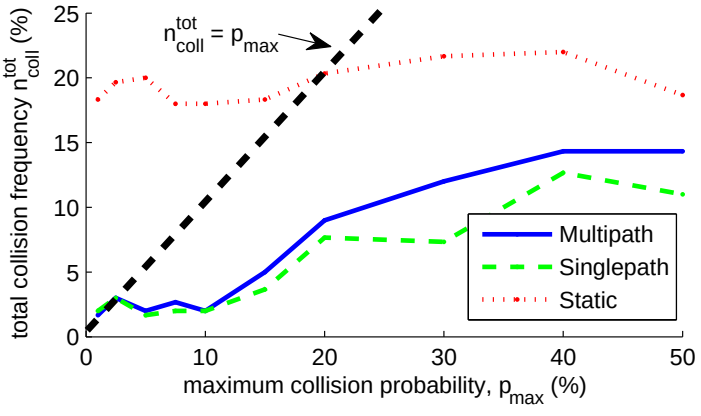


Figure 3.15: Total collision frequency, $n_{\text{coll}}^{\text{tot}}$, as a function of the collision probability bound, $p_{\max}$.

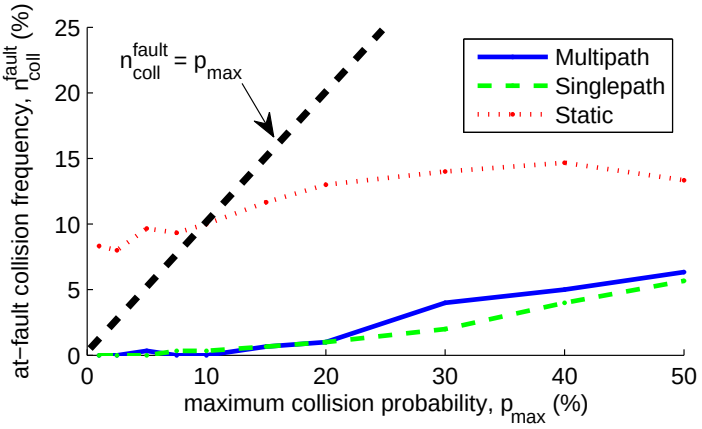


Figure 3.16: At-fault collision frequency, $n_{\text{coll}}^{\text{fault}}$, as a function of the collision probability bound, $p_{\max}$.

Compared with the other planners, the collision frequency of the Static planner is less sensitive to changes in $p_{\max}$. Since the Static planner is unable to anticipate obstacle motion, it encounters a high number of unavoidable collision scenarios, such as merging into an obstacle or being cut off or hit in the side/back by the obstacle; these cases are uncorrelated with $p_{\max}$. Figure 3.16 shows a similar general trend for at-fault collision frequencies, with the notable difference that the

Singlepath and Multipath planners are effectively collision-free ($n_{\text{coll}}^{\text{fault}} \leq 0.33\%$) for $p_{\max} \leq 10\%$. Additionally, the Static planner still experiences a large number of at-fault collisions over the entire range of $p_{\max}$ values.

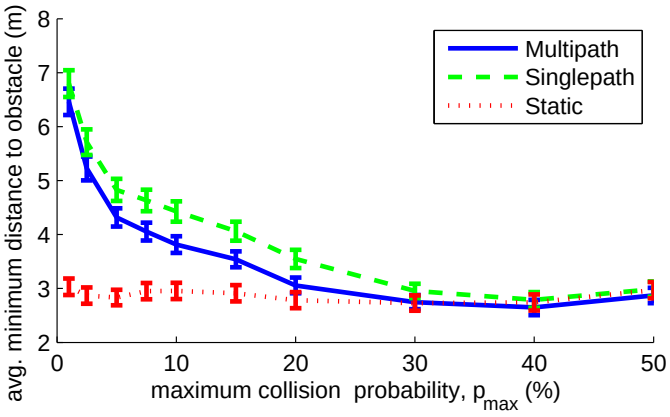


Figure 3.17: Average minimum distance to obstacle as a function of the collision probability bound, $p_{\max}$. Error bars indicate the standard error of the mean.

Figure 3.17 shows how the average minimum distance to obstacle metric varies as a function of $p_{\max}$. At very low values of $p_{\max}$ ($< 3\%$), the performance of the Multipath planner matches that of the Single path planner. As $p_{\max}$ increases, both the Singepath and the Multipath planners approach the performance of the Static planner; however, the Multipath planner does so at a faster rate. These results are intuitive, since at low values of $p_{\max}$, the Multipath planner attempts to avoid any possible obstacle trajectory, causing the robot to behave as conservatively as the Singlepath planner. At mid to high $p_{\max}$ values, the Multipath planner is able to accept some collision risk and actively utilizes its contingency plans to improve performance while maintaining low collision frequencies. Finally, at very high $p_{\max}$ values, collision risk is relatively unimportant and the Singlepath planner approaches the aggressiveness of the other planners.

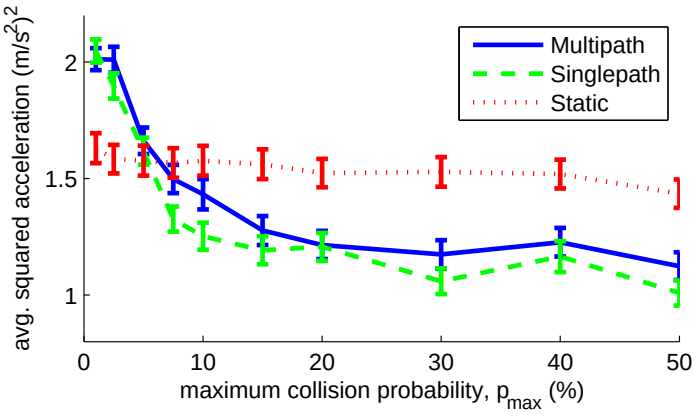


Figure 3.18: Average squared acceleration as a function of the collision probability bound, $p_{\max}$. Error bars indicate the standard error of the mean.

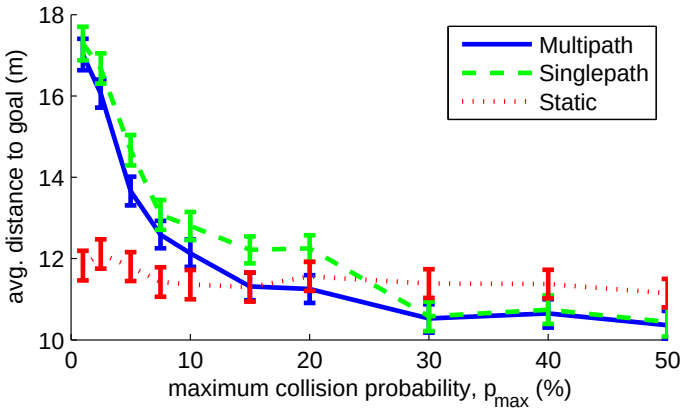


Figure 3.19: Average distance to goal as a function of the collision probability bound, $p_{\max}$. Error bars indicate the standard error of the mean.

Figures 3.18-3.19 plot the average squared acceleration and distance to goal metrics as a function of $p_{\max}$. Both of these plots show a similar trend: at very low values of the $p_{\max}$ ($< 3\%$), the Multipath planner matches the Singlepath planner since both planners avoid any potential collision scenario. As $p_{\max}$ increases, the Singlepath and the Multipath planners diverge to a small degree,

and both approach, then exceed the performance of the Static planner. The acceleration results in Figure 3.18 suggest that the aggressiveness of the Multipath planner causes a net loss in acceleration performance compared to the Singlepath planner over most of the tested $p_{\max}$ range. The Multipath planner allows for smaller accelerations when the robot correctly recognizes that its path is clear before entering the intersection. However, due to its increased aggressiveness, it also experiences larger decelerations in cases where the robot must stop to avoid an obstacle. For the distance to goal metric, the Multipath planner holds a noticeable advantage over the Singlepath algorithm for the range from $p_{\max} = 5\%$ to $p_{\max} = 30\%$. This advantage is due to the more aggressive driving style of the Multipath planner. For $p_{\max} > 30\%$, collision probability becomes less important and the Singlepath Planner again approaches the performance of the Multipath planner.

3.5.3 Multiple Obstacle Simulation Results

Simulations for two scenarios were run to evaluate scaling, performance, and the affects of traffic density in multiple obstacle encounters. These scenarios were conducted using the obstacle trajectory clustering algorithm proposed in Section 3.4.1. Both scenarios consist of a robot and two obstacle vehicles navigating the intersection scenario depicted in Figure 3.20. The first set of simulations represents a congested scenario, where the robot and two obstacle vehicles all reach the intersection at approximately the same time; the interior distributions in Figure 3.20 depict the possible initial conditions for the robot and both obstacle vehicles for these congested simulations. The second set of simulations represents a less-congested scenario, where one obstacle reaches the intersec-

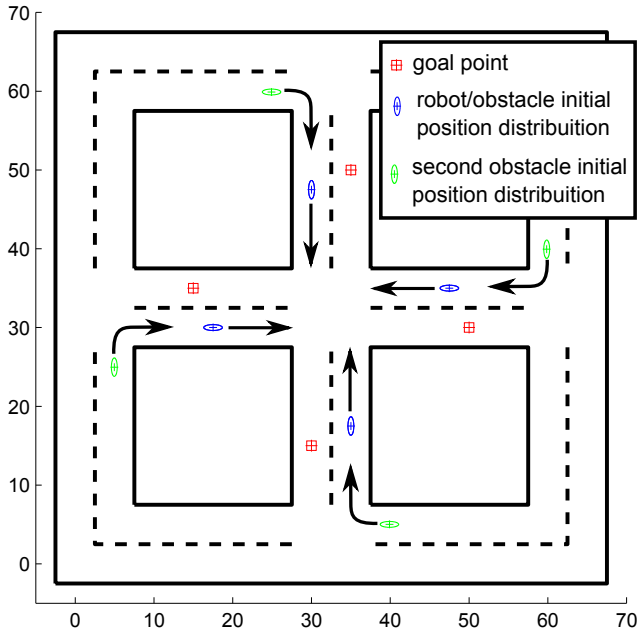


Figure 3.20: Possible initial conditions for the multiple-obstacle simulations. For congested simulations, the robot and both obstacles are randomly assigned initial positions from the interior distributions. For less-congested simulations, one obstacle is initialized using one of the exterior distributions.

tion around the same time as the robot, but the other obstacle arrives later. The first obstacle is randomly assigned an initial condition from the interior set of distributions, as in the congested case, while the second obstacle is initialized using one of the exterior distributions, as shown in Figure 3.20. Both sets of simulations included $n_{\text{sim}} = 500$ runs, each for 8 seconds, using maximum cluster values ranging between $n_c = 1$ and $n_c = 4$. The Static planner was also tested to provide a baseline comparison. The Singlepath planner from the previous section is equivalent to the $n_c = 1$ case.

Table 3.3: Performance metrics for $n_{\text{sim}} = 500$ congested simulation runs

			min. distance to obstacles		avg. squared acceleration		min. distance to goal	
n_c	$n_{\text{coll}}^{\text{tot}}$	$n_{\text{coll}}^{\text{fault}}$	μ_{obst}	SE_{obst}	μ_{acc}	SE_{acc}	μ_{goal}	SE_{goal}
1	2.0	0.0	2.59	0.09	1.80	0.048	14.44	0.22
2	1.4	0.0	2.55	0.09	1.86	0.049	14.27	0.24
3	2.0	0.0	2.54	0.09	1.93	0.050	14.21	0.23
4	1.4	0.0	2.53	0.09	1.97	0.051	14.15	0.23
Static	30.2	11.4	1.69	0.08	2.26	0.46	13.45	0.27
units	%	%	(m)		$(\text{m/s}^2)^2$		(m)	

Congested Intersection

The congested simulations challenge the clustering algorithm since all predicted obstacle trajectories for both obstacles are potentially important and influence the robot over a similar time period. Table 3.3 presents the minimum distance to obstacle, average squared acceleration, and distance to goal performance metrics evaluated over all $n_{sim} = 500$ congested simulations. The total collision frequency, $n_{\text{coll}}^{\text{tot}}$, and total at-fault collision frequency, $n_{\text{coll}}^{\text{fault}}$, are also presented.

All metrics in Table 3.3 show clear trends as n_c varies: the minimum distance to obstacle and distance from goal metrics get smaller as n_c increases, while the average squared acceleration metric grows larger. The reason for these trends is that using multiple contingency paths allows the robot to delay making decisions, such as braking for obstacle avoidance, while there is still ambiguity

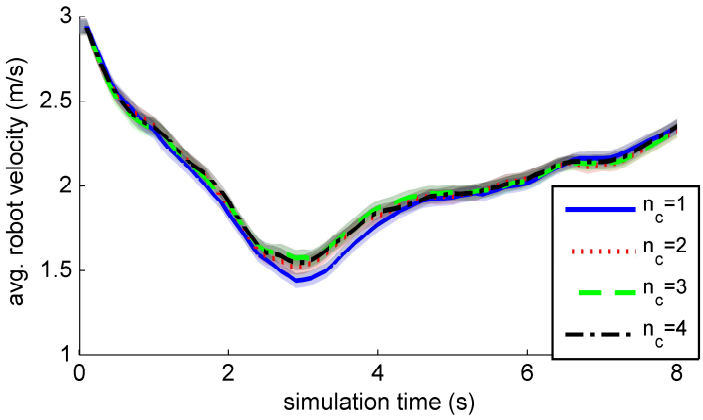


Figure 3.21: Average velocity as a function of simulation time for the congested simulations. Shaded regions reflect the one standard error of the mean (SE_{vel}) uncertainty bounds.

in an obstacle's goal and while there is still time for the robot to react safely to any possible outcome. Since the robot has a small likelihood of being able to navigate the congested intersection without stopping, statistical differences in robot's performance as a function of n_c are small. If, as in this scenario, the robot delays braking but typically must stop and wait, the delayed braking decision causes the robot to 1) stop closer to the obstacle and 2) brake more aggressively. This ability to delay the braking decision also statistically improves the robot's performance with respect to the distance to goal metric. This delayed braking is visible in the $t = 2$ to $t = 4$ second range of Figure 3.21, which shows the velocity history of the robot averaged over all $n_s = 500$ simulations. Despite this delayed decision making, the contingency planner, at all n_c values, has a significantly lower collision frequency than the Static planner, and increasing n_c appears to have no correlation with an increase in collision frequency. Table 3.4 shows the paired *t*-test results for each metric, evaluating whether differences in the metric between the $n_c > 1$ cases and the $n_c = 1$ case are statistically significant. These tests indicate that the distance to obstacle and average squared

Table 3.4: Paired *t*-test results for congested case at $\alpha = 0.05$

	min. distance to obstacle		avg. squared acceleration		min. distance to goal	
comparison	H	P	H	P	H	P
$n_c = 2$ vs. $n_c = 1$	1	0.018	1	0.009	0	0.153
$n_c = 3$ vs. $n_c = 1$	1	0.004	1	< 0.001	0	0.058
$n_c = 4$ vs. $n_c = 1$	1	0.006	1	< 0.001	1	< 0.001

acceleration metrics show a statistically significant change in performance over the entire range of n_c, while the distance to goal metric only becomes statistically significant for the $n_c = 4$ case.

Less Congested Intersection

The less-congested simulations allow for more obvious obstacle trajectory clusterings since one obstacle's trajectories are clearly more important than those of the other. These simulations are also less challenging for contingency planner since the dynamic obstacles are more spread out both spatially and temporally. Table 3.5 presents the performance metrics for the less-congested simulations. The results show similar trends as in the congested simulations, with the minimum distance to obstacle and distance from goal metrics getting smaller as n_c increases, and the average squared acceleration metric growing larger. However, the changes in the minimum distance to obstacle and distance to goal metrics are more pronounced while the changes in the average squared acceleration metric are nearly the same. Since the robot is less likely to require complete

Table 3.5: Performance metrics for $n_{\text{sim}} = 500$ less-congested simulation runs

			min. distance to obstacles		avg. squared acceleration		min. distance to goal	
n_c	$n_{\text{coll}}^{\text{tot}}$	$n_{\text{coll}}^{\text{fault}}$	μ_{obst}	SE_{obst}	μ_{acc}	SE_{acc}	μ_{goal}	SE_{goal}
1	2.2	1.2	3.10	0.094	1.82	0.055	14.53	0.300
2	1.4	0.8	2.95	0.086	1.87	0.058	13.95	0.299
3	1.8	0.8	2.86	0.086	1.98	0.062	13.71	0.297
4	1.4	0.6	2.89	0.083	1.97	0.061	13.66	0.301
Static	23.2	13.4	1.99	0.083	1.68	0.050	11.39	0.284
units	%	%	(m)		$(\text{m/s}^2)^2$		(m)	

stops than in the congested simulations, it is better able to utilize available contingency plans to improve performance. Additionally, there is negligible difference between the $n_c = 3$ and $n_c = 4$ cases, reflecting the fact that the robot primarily must consider only one important obstacle at a time, reducing the number of important obstacle prediction clusters to three. Figure 3.22 shows the average velocity profiles for the less-congested simulations. These velocity profiles show that 1) the robot requires less braking to navigate the intersection compared to the congested simulations (Figure 3.21) and 2) there is a significant difference between the conservativeness of the $n_c = 1$ case and the $n_c > 1$ cases.

Table 3.6 shows the paired *t*-test results for the less-congested simulations. These tests show an increase in the statistical significance of the distance to ob-

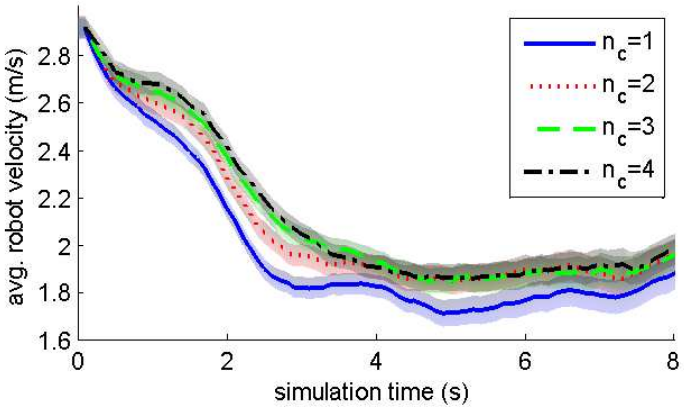


Figure 3.22: Average velocity as a function of simulation time for the less-congested simulations. Shaded regions reflect the one standard error of the mean (SE_{vel}) uncertainty bounds.

Table 3.6: Paired *t*-test results for less-congested case at $\alpha = 0.05$

	min. distance to obstacle		avg. squared acceleration		min. distance to goal	
comparison	H	P	H	P	H	P
$n_c = 2$ vs. $n_c = 1$	1	< 0.001	0	0.089	1	< 0.001
$n_c = 3$ vs. $n_c = 1$	1	< 0.001	1	< 0.001	1	< 0.001
$n_c = 4$ vs. $n_c = 1$	1	< 0.001	1	< 0.001	1	< 0.001

stacle and distance to goal metrics, while the significance of the average squared acceleration metric is similar to the congested case.

Table 3.7: Computation time statistics

	single obstacle		multi-obstacle congested		multi-obstacle less-congested	
n_c	μ_{comp}	σ_{comp}	μ_{comp}	σ_{comp}	μ_{comp}	σ_{comp}
1	0.035	0.14	0.21	0.54	0.38	0.73
2	-	-	0.31	0.77	0.47	0.89
3	0.054	0.14	0.44	1.02	0.57	1.06
4	-	-	0.53	1.23	0.62	1.16
Static	0.032	0.07	0.23	0.39	0.18	0.34
units	s	s	s	s	s	s

3.5.4 Computational Performance

Table 3.7 shows computation time statistics for each of the presented simulation scenarios. The computation time mean, μ_{comp} and standard deviation, σ_{comp}, are included for each value of n_c and for the static path planner. For the single obstacle case, the Multipath planner experiences a maximum of $n_s = 3$ obstacle predictions and is equivalent to setting $n_c = 3$. Similarly, the Singlepath planner is equivalent to $n_c = 1$. The single obstacle results show that the Multipath planner is capable of real-time performance for up to three simultaneous contingency paths, with nominal rates greater than 10Hz.

The multiple obstacle computation time results in Table 3.7 show that real-time contingency planning for $n_c \geq 4$ is difficult, and that the exponential scaling of exhaustive contingency planning is not a feasible option. In both the

congested and less-congested scenarios the mean computation time varies approximately linearly with n_c. The slight increase in average computation time for the less-congested simulations relative to the congested simulations is due to the more spaced out obstacles in the less-congested simulations forcing the robot to perform non-trivial path planning over a longer portion of the simulation time window. The clustering algorithm was written in Matlab and took between 0.1 and 0.45 seconds. The dominant computational cost in the clustering algorithm is the driving corridor inclusion evaluation, which is not dependent on the maximum number of clusters n_c and grows linearly with the number of obstacle predictions.

3.6 Conclusions

A novel contingency path planning algorithm is presented which provides a framework for the inclusion of anticipated, probabilistic obstacle motion in the planning process. This planner uses a novel path optimization algorithm to simultaneously optimize multiple continuous contingency paths with a shared initial segment, allowing the robot to account for multiple mutually exclusive obstacle predictions while maintaining a deterministic path to execute at the current timestep. The path optimization formulation uses a spline based contingency path representation, and associated cost and constraint definitions, to enable fast optimization of the continuous contingency paths. A novel collision probability bound is also proposed which enables efficient computation of a tight bound on point-wise collision probability between oriented polygons with uncertain relative positions and orientations. This tight bound is critical to prevent over-conservative behavior in common road driving scenarios. Addi-

tionally, an algorithm for clustering obstacle trajectories is presented to enable this contingency planning approach to scale to more complex, multi-obstacle environments. Simulation results show that the presented Multipath algorithm is as safe in terms of collision frequency as the more conservative Singlepath approach, but is more aggressive in terms of obstacle spacing, and is faster through intersections. These results also hold for a wide range of values for the collision probability threshold, $p_{\max}$. Similar performance trends occur when using trajectory clustering in multiple obstacle scenarios. As the number of allowed clusters, n_c, increases, the robot experiences increases in aggressiveness in terms of obstacle spacing and speed through intersections, with no associated increase in collision probability.

Future goals for this research focus on improving models for obstacle prediction, in order to allow the proposed contingency planning framework to be applied to a wider array of real world driving scenarios. Model improvements being investigated include 1) learning obstacle behavior from previous interactions, 2) modeling additional obstacle types such as pedestrians and bicyclists, and 3) using both formal and social traffic rules to identify a broader set of possible obstacle intents.

Acknowledgments

This research is supported by ARO Grant #W911NF-09-1-0466, with Dr. Randy Zachery as Program Manager and was made with Government support under and awarded by DoD, Air Force Office of Scientific Research, National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFR 168a.

CHAPTER 4

EXPERIMENTAL EVALUATION OF A CONTINGENCY BASED PATH PLANNER FOR AUTONOMOUS DRIVING

Abstract

This paper presents simulation and experiment results for a contingency based path planner used for probabilistic collision avoidance in an autonomous road vehicle interacting with human driven obstacle vehicles at an uncontrolled four-way intersection. Tested scenarios are designed to consistently force the autonomous vehicle into dangerous collision avoidance interactions. A system for integrating virtual obstacle vehicles is used to safely test live collision avoidance interactions. The contingency planning approach is compared with a non-contingency approach as well as with the path planner used by Cornell in the 2007 DARPA Urban Challenge. Implementation challenges and limitations of the contingency planning approach are also discussed.

4.1 Introduction

Robustly adapting to and interacting with dynamic obstacles is a critical component in achieving safe, reliable autonomous driving systems. Autonomous road vehicles must be able to safely handle a wide range of obstacle behaviors and traffic types. Recent autonomous driving experiments such as the 2007 DARPA Urban Challenge have demonstrated that reactionary planning systems and deterministic obstacle handling logic can provide reasonable driving performance

when obstacles are well behaved and strictly follow predefined traffic rules [10]; however, these approaches break down and can lead to dangerous interactions when the autonomous vehicle experiences unusual obstacle behavior or during complex, unplanned interaction scenarios [20].

Dynamic obstacle uncertainty can be mitigated to some degree using networked communication, either through intelligent convoys [9][71], or through cooperative communication between intelligent vehicles [43][39]. For autonomous driving in the presence of dynamic obstacles with uncertain intentions, probabilistic obstacle anticipation models can provide critical information for planning and collision avoidance. Obstacle predictions based on inferred obstacle goals have been applied to mobile robots [63], autonomous road vehicles [17] [3], and air traffic control systems [37]. Similar anticipation models have also been developed for predicting the motion of pedestrians [88].

This paper presents simulation and experiment results that explore the effectiveness of incorporating anticipated obstacle motion into the path planning and decision making process, as well as the benefits and limitations of planning multiple continuous contingency paths to explicitly account for the possible mutually exclusive behaviors and maneuvers a dynamic obstacle might take.

The rest of the paper is organized as follows. Section 4.2 provides an overview of Cornell's autonomous vehicle platform Skynet. Section 4.3 presents a human-in-the-loop virtual obstacle interface which enables the safe testing of dangerous collision avoidance scenarios. Section 4.4 presents a general overview of the simulation and experiment setup. Section 4.5 presents live experiment results using a single human driven obstacle vehicle. Section 4.6 presents simulation results using prerecorded obstacle trajectories. Section 4.7

presents simulation results for both single and multiple human driven obstacles and Section 4.8 provides conclusions.

4.2 Autonomous Vehicle System Overview

Cornell's autonomous vehicle platform Skynet is an autonomous 2007 Chevrolet Tahoe. The basic architecture of Skynet's perception and planning systems is shown in Figure 4.1.

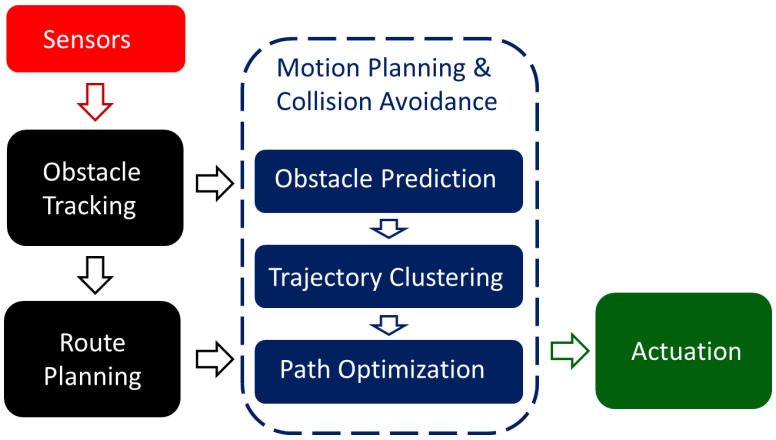


Figure 4.1: Block diagram of the contingency planning procedure.

A detailed overview of Skynet's Sensors, Obstacle Tracking, and Route Planning systems can be found in Miller et al. [60]. In depth coverage of the dynamic contingency planning approach used in the Motion Planning and Collision Avoidance block can be found in Hardy and Campbell [32]. The system details presented here are intended to provide context for the experimental results presented in this paper. Section 4.2.1 provides an overview of the Sensor block, as well as the computation platform and mechanical actuation systems on Skynet. Section 4.2.2 provides an overview of the Obstacle Tracking and Route

Planning blocks. Section 4.2.3 provides an overview of the dynamic contingency planning approach used for motion planning and collision avoidance.

4.2.1 Sensors, Computation, and Actuation

Skynet's perception system is designed to localize Skynet in a global reference frame and to track static and dynamic obstacles in a relative reference frame. Localization is achieved using GPS, IMU, and vehicle odometry measurements. An overview of Skynet's tightly-coupled pose estimation solution can be found in Miller et al. [62]. Skynet senses static and dynamic obstacles in its environment using a combination of laser rangefinders, radar, and vision cameras.

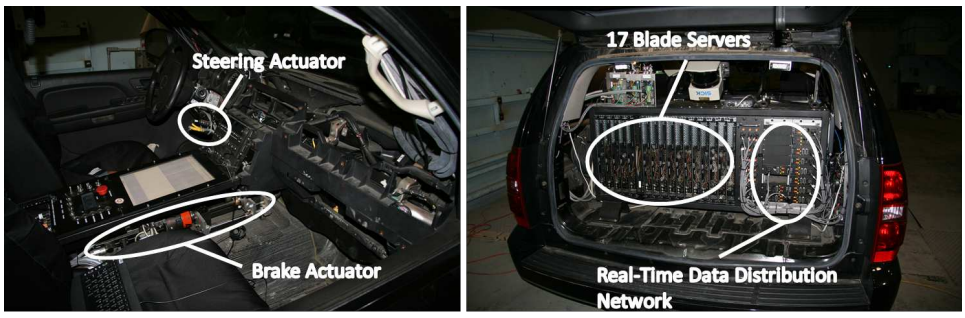


Figure 4.2: Mechanical actuators for brake and steering commands (left). Computation and dedicated timing microcontrollers (right).

The perception system avoids potential Windows timing and thread scheduling problems by accurately timestamping each sensor measurement using a dedicated microcontroller. This allows each estimation algorithm to maintain correct temporal ordering of data while tolerating common deviations in thread scheduling and execution times. Driving commands are executed on the vehicle using a custom steering, brake, and transmission actuation system.

Figure 4.2 shows the mechanical steering and brake actuators as well as the computer housing and the sensor microcontrollers.

4.2.2 Obstacle Tracking and Route Planning

The Obstacle Tracking and Route planning blocks are preprocessing steps that detect, identify and track dynamic obstacles and create high level navigation and behavior goals respectfully. The Obstacle Tracking block is responsible for identifying unique dynamic obstacles, tracking these obstacles between sensor measurements, and estimating their continuous state. Tracking is performed using a Rao-Blackwellized particle filter which uses a particle filter to solve the data association problem and a bank of extended Kalman filters (EKFs) to estimate the continuous state of each dynamic obstacle. A detailed overview of this obstacle tracking algorithm can be found in Miller et al. [61]. The output of this tracking algorithm is a set of Gaussian state estimates for each dynamic obstacle in the robot's environment.

The Route Planning block is responsible for planning an optimal route along the robot's *a priori* road network map. This high-level planning process also serves as finite state machine controlling transitions between driving behaviors such as stay-in-lane, parking, and blockage recovery. The output of the Route Planning block is a set of navigational waypoints defining a nominal path for the robot over a short time horizon.

4.2.3 Motion Planning and Collision Avoidance

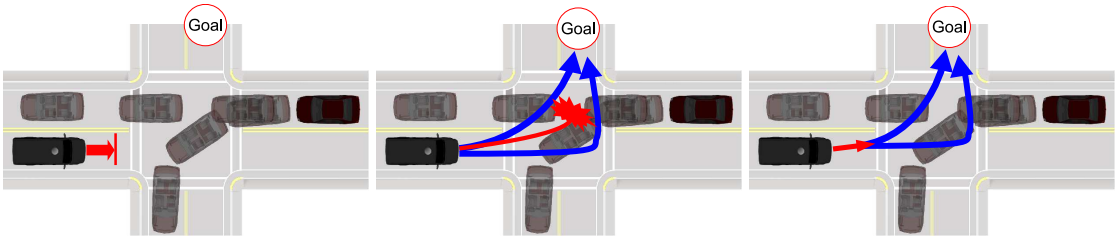
Dynamic obstacles identified and tracked in the Obstacle Tracking block not only have uncertainty in their current state due to sensor error and data association errors, they also have large uncertainties in their intent and in their future state trajectory. Popular approaches to predicting the motion of dynamic obstacles include using constant velocity and constant curvature models such as in [47], or predicting an obstacle along an assumed path either from *a priori* information about the environment or from an environmental decomposition such as a tangent graph [63]. However, these approaches oversimplify the problem and are insufficient for capturing complex, goal oriented obstacle behavior.

For the specific case of autonomous road vehicles, it is possible to utilize the known structure of the environment to make assumptions about the possible goals of a dynamic obstacle. The Tartan Racing team implemented a prediction strategy on their DUC robot which made deterministic trajectory predicts for each dynamic obstacle in the scene along every reachable path on the road network over a limited planning horizon [17]. This approach allows the robot to model multi-modal obstacle intent, however the deterministic nature of the obstacle predictions in [17] makes the robot's collision probability assumptions overconfident.

In previous work, [32], an obstacle prediction strategy is adopted where multiple probabilistic motion predictions are made for each dynamic obstacle based on their possible discrete goal modes as determined by an *a priori* map of the road network.

Here, uncertainty in the future state evolution of the robot's environment

is encoded as a set of mutually exclusive obstacle predictions. In this framework, a separate contingency path is planned for each possible permutation of predicted obstacle trajectories. The approach used here further constrains these contingency paths to share the same initial segment. By using a single shared path segment at the start of the path, the Contingency Planner can provide the robot with an immediate deterministic action. Multiple distinct contingencies are maintained for the rest of the path to account for uncertainty in the evolution of the robot's dynamic environment.



(a) Single Path (b) Weighted Combination (c) Shared Segment

Figure 4.3: A comparison of different strategies for path planning using mutually exclusive sets of obstacle predictions.

This partially shared contingency planning approach is preferable to alternate approaches, such as planning a single path which attempts to avoid all obstacle predictions simultaneously as shown in Figure 4.3(a), or taking a weighted combination of independent contingency paths as shown in Figure 4.3(b). Planning a single path is overcautious and ignores the fact that the obstacle goal sets are mutually exclusive, while taking a weighted combination of independent contingency paths offers no guarantees on the safety of the combined path. Sharing the initial segment of each contingency path, as shown in Figure 4.3(c), gives the robot a deterministic action to execute at the current time step while maintaining the independence of future contingencies. The proposed contingency planning procedure is shown in Figure 4.1. The trajectory cluster-

ing algorithm functions as a preprocessing step to reduce the complexity of the obstacle predictions prior to contingency planning.

Obstacle Prediction

Each obstacle trajectory prediction for a given goal mode is initialized using the obstacle state estimate at the current timestep and propagated forward in time using a probabilistic motion model. This motion model can vary in complexity from a simple path following model that tracks the center of the driving lane, to a more complex path planner tuned to each obstacle type. For linear models, this propagation can be performed using the prediction step of the Kalman Filter and for nonlinear, non-differentiable motion models, this propagation can be performed using an algorithm such as the Unscented Transform [42].

For the simulations and experiments presented in this paper, a simple pure pursuit path following model [12] with linear velocity feedback control was used for the obstacle prediction model. This model is used for two reasons: 1) to demonstrate that even low fidelity obstacle prediction models can significantly improve planning robustness and reliability and 2) to enable real time implementation of the Motion Planning and Collision Avoidance layer.

The output of this obstacle prediction for the i^{th} dynamic obstacle is a set of predicted obstacle trajectories, $\mathcal{T}_i$. Each trajectory prediction, $T \in \mathcal{T}_i$, is represented as a sequence of obstacle state distributions over N timesteps into the future:

$$T = \{(\mu_i, \Sigma_i), (\hat{\mu}_i, \hat{\Sigma}_i)_1, \ldots, (\hat{\mu}_i, \hat{\Sigma}_i)_N\} \tag{4.1}$$

where (μ_i, Σ_i) represents the current state estimate of obstacle i and $(\hat{\mu}_i, \hat{\Sigma}_i)_k$

represents a predicted obstacle state distribution at future timestep k.

For multiple dynamic obstacles, the set of all possible permutations of predicted obstacle trajectories is defined as:

$$\mathbf{T} = \mathcal{T}_1 \times \mathcal{T}_2 \times \ldots \times \mathcal{T}_{n_o} \tag{4.2}$$

where $\mathbf{T}$ is the Cartesian product of the predicted obstacle trajectory sets for all n_o obstacles. The total number of possible obstacle trajectory permutations in the environment, n_s, is defined as:

$$n_s = \prod_{i=1}^{n_o} |\mathcal{T}_i| \tag{4.3}$$

Trajectory Clustering

The goal of contingency planning is to generate a set of planned paths that account for all possible evolutions of the robot's environment. In this framework, a separate contingency path is planned for each possible permutation of predicted obstacle trajectories. Equation 4.3 shows that the number of possible permutations, n_s, scales exponentially with the number of obstacles, n_o, and the number of trajectory predictions for each obstacle, $|\mathcal{T}_i|$. This exponential scaling in the number of required contingency paths can be avoided by clustering the obstacle prediction permutations prior to executing the Contingency Planner based on similarities in their potential influence on the robot's future path. This allows the Contingency Planner to use a fixed maximum number of contingency plans, n_c, regardless of the number of obstacles and possible obstacle goals in the environment. Clustering similar predicted obstacle trajectory permutations provides improved computational scaling because it allows the planner to ignore combinatorial permutations of obstacle predictions within trajectory clus-

ters, under the assumption that these permutations all have a similar influence on the robot's future path and can be safely approximated as simultaneously occurring events.

Path Optimization

The contingency planning problem is formulated as a constrained optimization problem and can be solved using well known convex optimization techniques, including interior point methods [21] and sequential quadratic programming methods [8]. The optimization cost function is defined based on desirable path goals and behaviors, and optimization constraints are defined to enforce safety requirements and to ensure conformance with vehicle limitations. Obstacle motion predictions are included in the path planning process both as a proximity bias in the path optimization objective function and as a constraint requiring all paths to satisfy a set maximum collision probability threshold.

4.3 Virtual Obstacle Integration

A virtual obstacle framework was developed in order to provide a safe framework for testing planning decisions during dangerous collision avoidance scenarios. An open source driving simulator was adapted to function as an external driving simulation environment to provide two-way interaction between Skynet and one or more human controlled virtual obstacle vehicles. The simulation environment used for this purpose is The Open Racing Car Simulator (TORCS) [16]. This virtual obstacle vehicle approach enables safe testing of dangerous collision avoidance scenarios while providing realistic two-way interac-

tions with human driven obstacle vehicles.

Figure 4.4 shows a block diagram of the Skynet planning architecture with the driving simulator interface. Virtual obstacle information from the driving simulation is received by the Obstacle Tracking block of Skynet's architecture and is appended to the list of tracked obstacles that is broadcast to the Route Planning and Obstacle Prediction blocks. The Obstacle Tracking block also broadcasts Skynet's current pose estimate to the external driving simulator, allowing the human obstacle driver to interact with Skynet in real time.

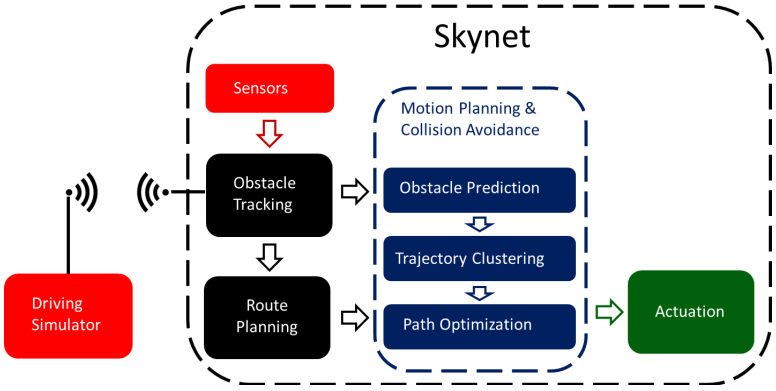


Figure 4.4: Block diagram of the Skynet architecture with the Driving Simulator as an external source of obstacle information.

Wireless communication between the driving simulator and Skynet is conducted over an existing wireless bridge that was designed to allow the Skynet chase vehicle to monitor Skynet's sensor inputs in real-time. Human drivers interface with the driving simulator using a Logitech G27 Racing Wheel peripheral which supplies both steering and pedal inputs. Figure 4.5 shows the driving simulator setup during an experiment.

Implementation challenges with the use of virtual obstacle vehicles include wireless communication limitations and driving realism and fidelity for the hu-

Figure 4.5: The driving simulator in use during an experiment.

man obstacle drivers. The limited range of the wireless bridge and need for continuous line-of-site between a base station and Skynet placed restrictions on the test site location. To meet these requirements, intersection testing was performed in an on-campus parking lot using a virtual intersection map. This allowed the base station to be placed close to the intersection and provided clear line-of-site for the entire testing area.

Driving realism was also a concern both because the driving simulation software was designed as a racing simulation and because the Logitech driving peripherals used lack realism compared to advanced driving simulators such as in [19]. To address these concerns, a custom car model was used that more closely approximates the power and performance of a standard production vehicle and each human driver was allowed to practice on the virtual test course until they felt comfortable controlling the obstacle vehicle. Despite these precautions, the human-in-the-loop experiments in Sections 4.5 and 4.7 experience a small num-

ber of instances where inadvertent over-steering and under-steering by virtual obstacle drivers caused minor collisions even when Skynet was behaving safely.

Along with providing real-time human driven obstacle interactions for Skynet in live testing situations, the driving simulator is also able to provide both prerecorded and real-time obstacle interactions through the Skynet simulation environment. Prerecorded human driven obstacle trajectories are used in Section 4.6 to provide consistent test conditions and enable large numbers of simulations. Simulations using human-in-the-loop obstacle interactions are used in Section 4.7 to extend the live experiment results presented in Section 4.5.

4.4 Experiment Setup

The experiment and simulation results presented in this paper involve Skynet tasked with navigating a four way intersection in the presence of one or more human driven obstacle vehicles with no stop signs or traffic rules. The human drivers are instructed to drive through the intersection under the assumption that they have the right-of-way. Skynet is then tasked with navigating the intersection while avoiding dynamic obstacles. These simulations represent a worst case intersection scenario and are designed to ensure efficient use of simulation runs by consistently forcing the robot into interesting obstacle avoidance interactions. Unlike the simulation results presented in [32], both the experiments and simulations presented here occur asynchronously to Skynet's planning and collision avoidance computations. Additionally, all dynamic obstacles are controlled by human operators, either in the form of prerecorded human controlled obstacle trajectory data, or in the form of a live human driver in a two-way driv-

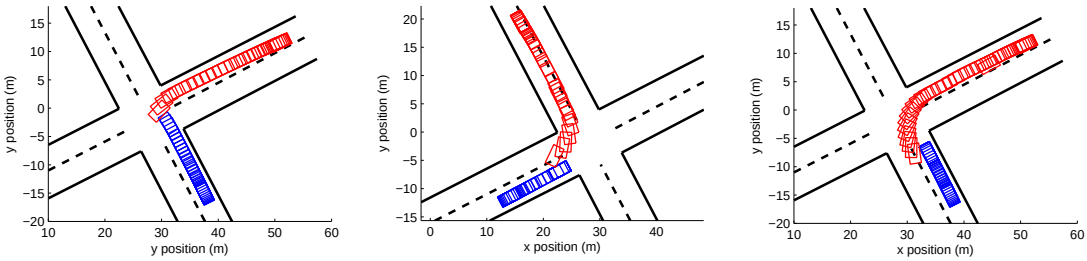
ing simulation.

Due to the condensed number of risky encounters, more aggressive braking and higher collisions frequencies are expected compared to normal driving. Collision frequency results are presented here as a relative metric of safety performance between path planning algorithms. As discussed in Section 4.2.3, the obstacle prediction model used in Skynet is only a rough approximation of human driven obstacle behavior. Additionally, in simulations using prerecorded obstacle trajectories, the obstacles behave with no regard for the autonomous vehicle since their trajectories are recorded off-line in an obstacle free environment.

All simulation and experimental results provide a comparison between three planning algorithms: 1) Multipath: the contingency approach proposed in [32] which plans separate contingency paths for n_c mutually exclusive clusters of obstacle prediction permutations, 2) Singlepath: a single path variation of the Multipath planner where a single contingency path is used that avoids all obstacle predictions (equivalent to $n_c = 1$), and 3) DUC: The path optimization algorithm used by Skynet in the DUC as described in [60],[33] where dynamic obstacles are treated as stationary. All other optimization parameters are identical. The error bounds associated with the average velocity and brake pressure results presented in this paper indicate one standard error of the mean, $SE = \frac{\sigma}{\sqrt{n_{\text{sim}}}}$, unless otherwise indicated. Simulations were run in the C# Skynet simulation environment and the path optimization solution was written in C++ using the open source nonlinear optimization library IPOPT [80].

4.4.1 Collision Classification

The driving controls used in the virtual obstacle simulation environment presented in Section 4.3 only provide a low fidelity simulation of real driving for the human operator. This lack of realism is minimized by allowing the human drivers to practice driving in the driving simulation environment. However, unfamiliarity with the simulation driving controls led to a small number of collision scenarios in which unintended understeer or oversteer by the human driver resulted in a collision between the human driven obstacle vehicle and the autonomous vehicle either as the autonomous vehicle was still approaching the intersection, or after the autonomous vehicle had safely stopped prior to entering the intersection.



(a) At-fault (b) Not-at-fault (Understeer) (c) Not-at-fault (Oversteer)

Figure 4.6: Collision classification scenarios between the autonomous vehicle (blue) and a human driven obstacle vehicle (red).

Figure 4.6 presents three scenarios from the human-in-the-loop simulations in Section 4.7. Figure 4.6(a) shows a typical collision where both the robot (blue) and the obstacle vehicle (red) are partially at fault. Figure 4.6(b) shows a not-at-fault collision where the robot stopped safely before the intersection but was still hit by the obstacle vehicle due to unintended understeer. Figure 4.6(c) shows a similar not-at-fault collision where the robot stopped safely before the intersec-

tion but was hit by the obstacle vehicle due to unintended oversteer.

Collision frequency statistics for both the experimental trials in Section 4.5 and the human-in-the-loop simulations in Section 4.7 are presented using these at-fault and not-at-fault labels. At-fault collisions should be considered cases where the robot was genuinely behaving in an unsafe manner. Not-at-fault collisions are strictly a result of the human obstacle driver failing to maintain their lane during a turn.

4.5 Experiments

This section presents experimental results for the autonomous vehicle system described in Section 4.2. Figure 4.7 shows the experiment setup, with tested Skynet maneuvers in blue and obstacle vehicle maneuvers in red. A total of 56 test runs were performed per planner using four human obstacle drivers. For these experiments, the MultiPath planner was allowed to plan up to three contingency paths to account for all possible obstacle trajectories.

Figures 4.8 and 4.9 present the average brake pressure and velocity of the robot for each of the three tested algorithms. These results indicate that the MultiPath planner is slightly more aggressive than the SinglePath planner, while both are significantly more conservative than the DUC planner. Figure 4.10 shows both how close the robot came to the obstacle for every simulation run and how fast the robot was traveling at this closest point. This plot provides an indication of the robot's aggressiveness in terms of obstacle spacing as well as an indication of how cautiously the robot behaves around dynamic obstacles. Points on the top left of the plot represent dangerous interactions because

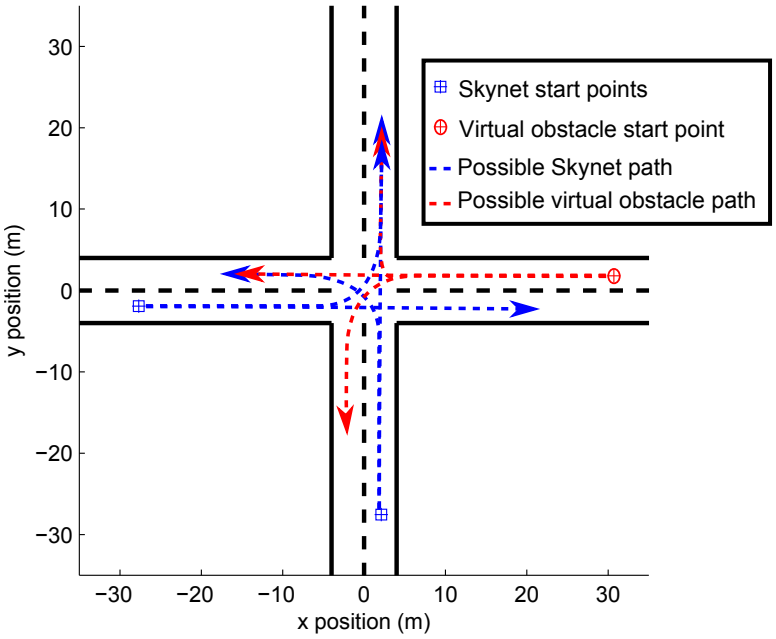


Figure 4.7: Experiment setup with possible robot maneuvers in blue and virtual obstacle maneuvers in red.

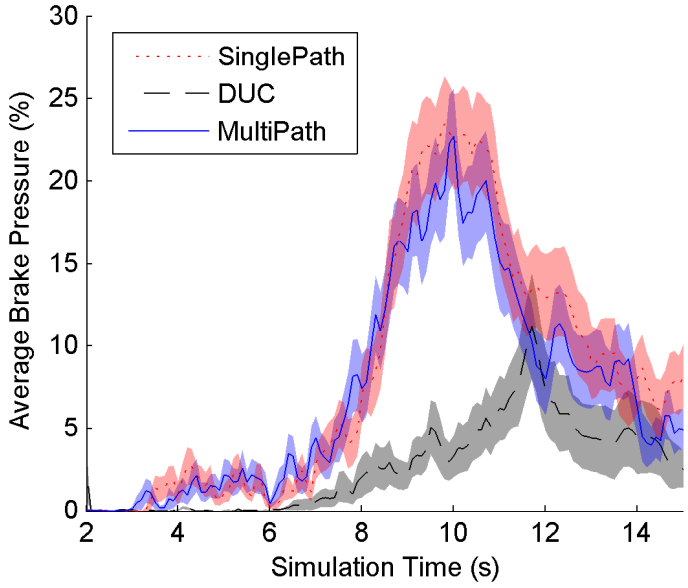


Figure 4.8: Average brake pressure as a function of experiment time.

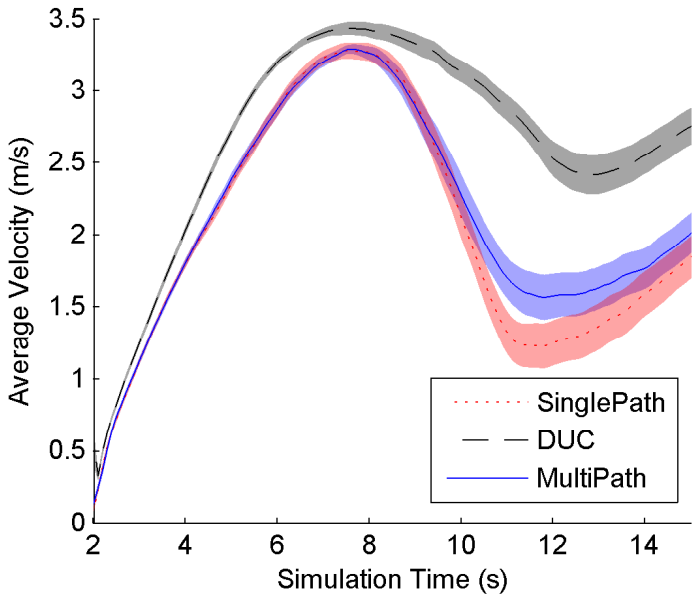


Figure 4.9: Average velocity as a function of experiment time.

they represent situations where the robot comes into close proximity with the dynamic obstacle while traveling at a relatively high rate of speed. The DUC planner interactions are clustered around this top left corner, indicating that the DUC planner experienced a much higher frequency of close calls and dangerous interactions than either the SinglePath or the MultiPath planners.

Skynet experienced one collision for each planner type during the experiments. Both the SinglePath and MultiPath planners experienced one not-at-fault collision while the DUC planner experienced one at-fault collision ($n_{\text{coll}}^{\text{fault}} = 1.8\%$). These collision scenarios are shown in Figure 4.11. The at-fault collision experienced by the DUC planner is consistent with the high rate of dangerous interactions shown in Figure 4.10.

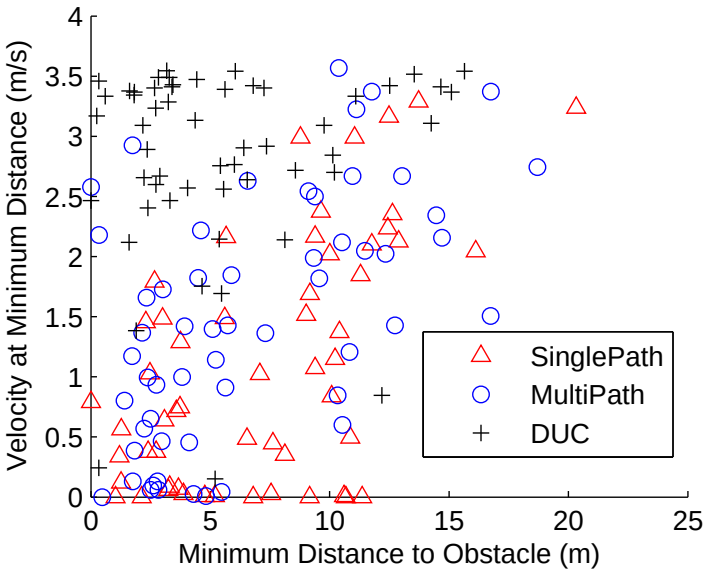
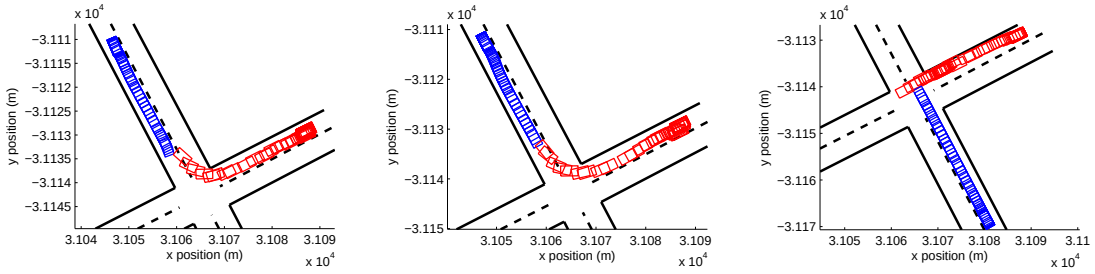


Figure 4.10: Velocity at minimum distance to obstacle for each experiment run.



(a) SinglePath Collision (not-at-fault)

(b) MultiPath Collision (not-at-fault)

(c) DUC Collision (at-fault)

Figure 4.11: Collisions experienced during live testing.

4.6 Simulation Results Using Prerecorded Obstacle Trajectories

This section presents simulation results from the Skynet simulator using prerecorded obstacle trajectories. The goal of these simulations is to provide a direct

comparison between the performance of the different planning algorithms using identical simulation conditions and a high volume of simulation runs. A total of 120 obstacle trajectories were recorded by four human drivers performing three distinct intersection maneuvers: turn left, drive straight, and turn right.

4.6.1 Single Obstacle Timing Sensitivity

To study the effects of timing on the relative planning performance of each algorithm, a single obstacle simulation was repeated with the robot starting at varying distances from the intersection. A total of 360 simulations were run for each starting distance. Figure 4.12 shows the possible simulation configurations. These simulations are designed to determine whether the performance advantages of the MultiPath planner compared to the SinglePath planner are dependent on what time the robot reaches the intersection relative to the obstacle vehicle. Figures 4.13 and 4.14 show the maximum average brake pressure and maximum average velocity, respectfully, as a function of the robot's starting distance from the intersection. The brake pressure results show that the MultiPath planner exhibits decreased breaking compared with the SinglePath planner over the entire range of tested starting distances. The velocity results show a similar trend except for the case where the robot starts at a distance of 18.4 meters from the intersection. In this case the SinglePath and MultiPath planners have similar peak velocities. Figure 4.15 shows the average velocity profiles for each planner at the 18.4 meter distance. These results show that while the MultiPath has a similar peak velocity as the SinglePath planner, it is able to maintain a higher average velocity after this point.

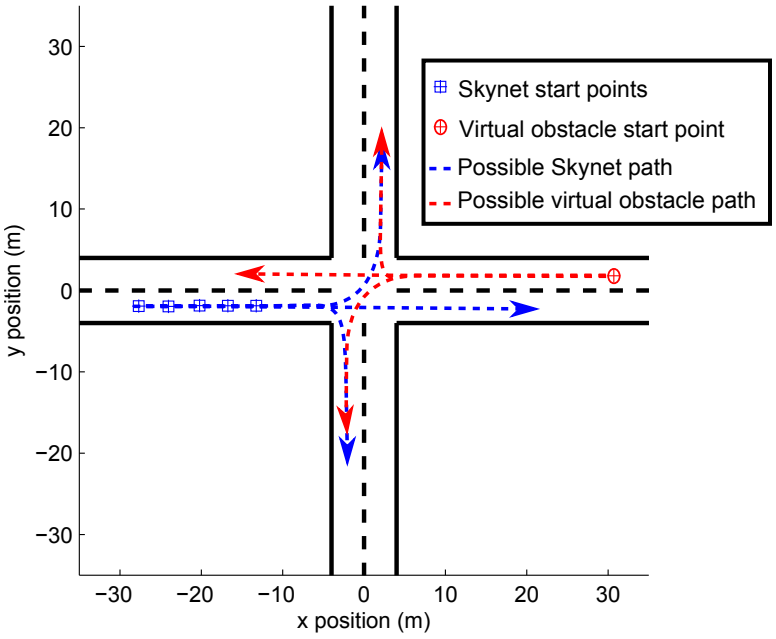


Figure 4.12: Simulation setup for timing sensitivity study with possible robot maneuvers in blue and virtual obstacle maneuvers in red.

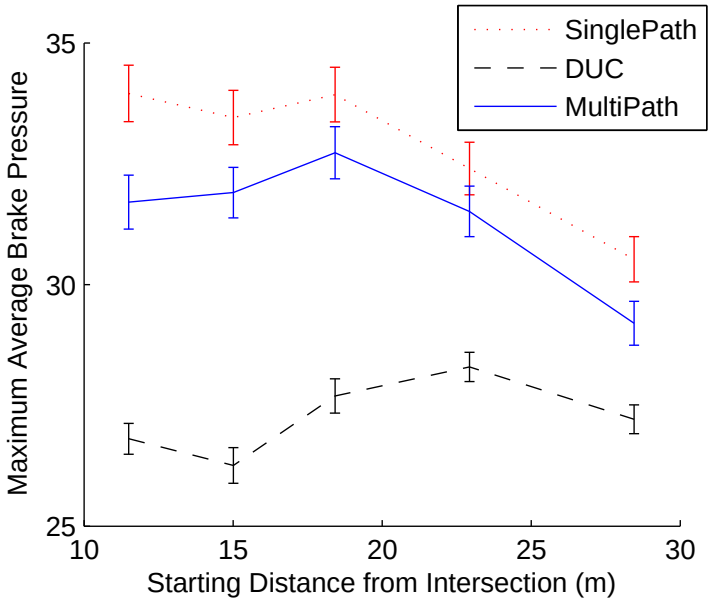


Figure 4.13: Maximum average brake pressure as a function of robot starting distance from intersection.

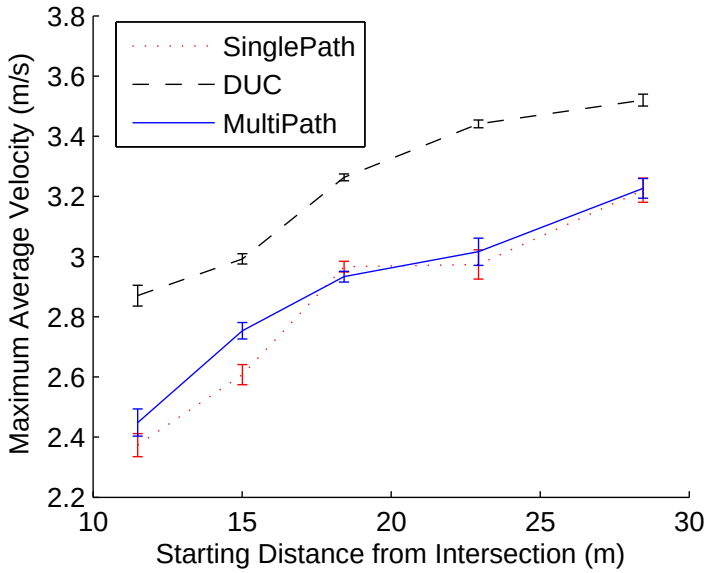


Figure 4.14: Maximum average velocity as a function of robot starting distance from intersection.

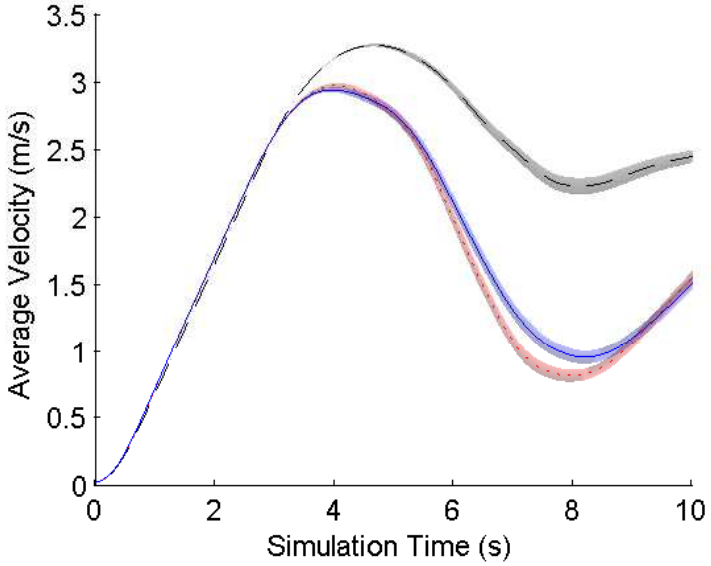


Figure 4.15: Average velocity as a function of simulation time for a starting distance of 18.4 meters from the intersection.

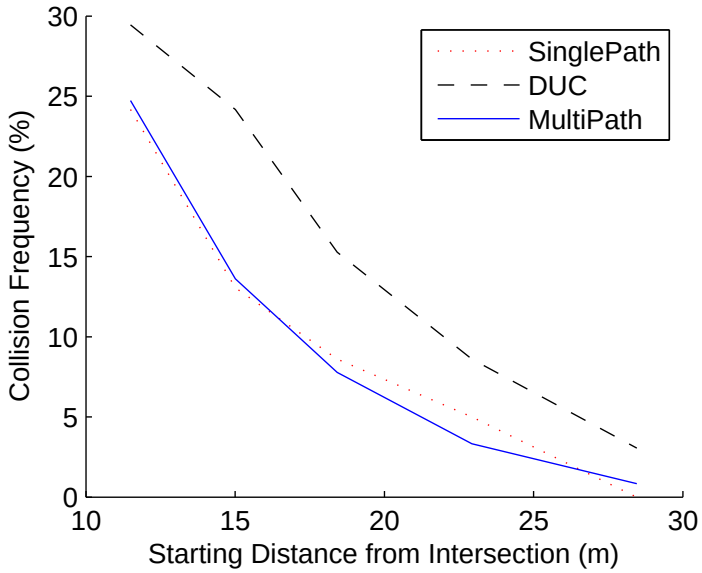


Figure 4.16: Collision frequency as a function of robot starting distance from intersection.

Figure 4.16 shows the collision frequency for each planner as a function of the robot's starting distance from the intersection. The use of prerecorded obstacle trajectories and low fidelity obstacle predictions mean that these collision results are not generalizable as a measure of real driving safety. However, these results show that the increased aggressiveness of the MultiPath planner is not associated with any increase in collision frequency relative to the SinglePath planner.

4.6.2 Multiple Obstacle Scaling

This section presents results for simulations with 3 simultaneous playback obstacles. These simulations are designed to assess the computational limitations of the contingency planning approach. Figure 4.17 shows the initial conditions

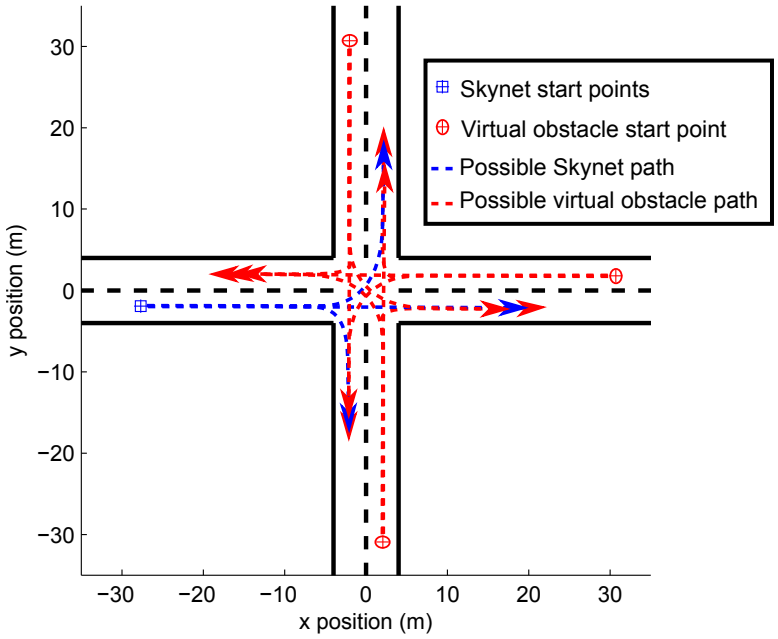


Figure 4.17: Simulation setup using 3 prerecorded obstacle drivers with possible robot maneuvers in blue and virtual obstacle maneuvers in red.

and possible maneuvers for each vehicle in the simulation. The prerecorded obstacle trajectories are rotated about the intersection center to provide obstacle data from three different directions. As the robot approaches the intersection, it must deal with 27 possible obstacle trajectory permutations. Trajectory clustering is performed to reduce the number of required contingency plans to a tested range of $n_c = [1, 4]$. The $n_c = 1$ case is equivalent to the SinglePath planner.

Figure 4.18 shows the computation time results for each of the tested values of n_c. The thick lines represent the mean path optimization time at each timestep, while the thinner lines indicate the upper bound of a two tailed 95% confidence region computed using an empirical cumulative distribution function. The path optimization was automatically terminated after a maximum computation time of 0.5 seconds. These results show that all tested values of

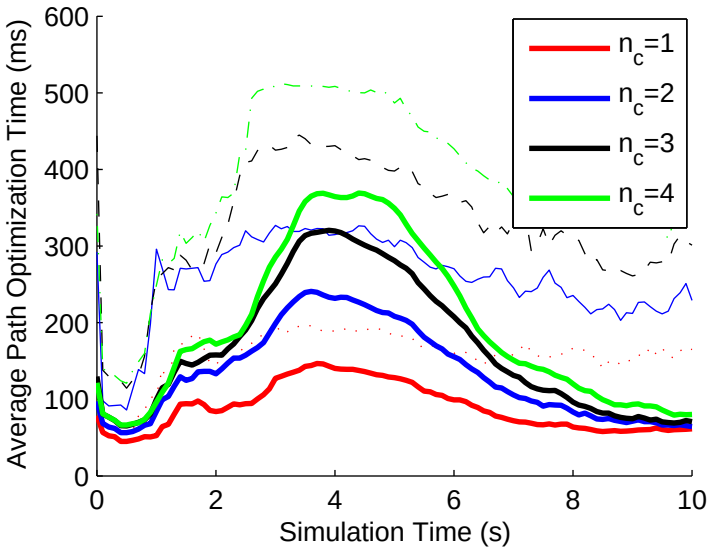


Figure 4.18: Average (bold) and two-tailed 95% confidence upper bound (thin) for path optimization time as a function of simulation time.

n_c were well within these computational limitations, though the $n_c = 4$ case likely experiences a small number optimization terminations due to excessive computation time.

Figure 4.19 shows the average brake pressure profile for each of the tested n_c values. These results show that increasing n_c beyond $n_c = 2$ allows the robot to delay braking as it approaches the congested 3-car intersection. The $n_c = 2$ case does not appear to provide any delay in braking compared to the $n_c = 1$ case, indicating that the trajectory clustering algorithm is unable to capture the most relevant mutually exclusive obstacle prediction information with a single additional contingency path.

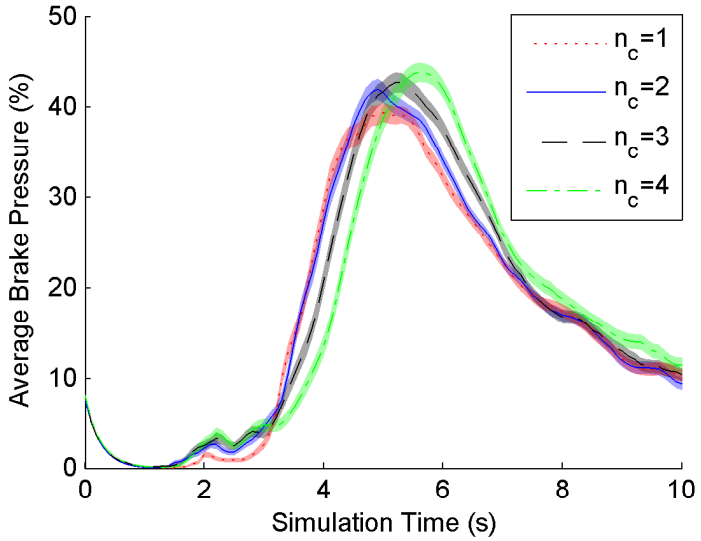


Figure 4.19: Average brake pressure as a function of simulation time.

4.7 Simulation Results Using Human Drivers

To extend the experimental results presented in Section 4.5, additional human-in-the-loop simulations were run using human driven obstacle vehicles in Cornell's autonomous driving simulation environment.

4.7.1 Single Human Obstacle Simulation Results

The experiment runs in Section 4.5 were recreated in simulation using four new human drivers. These simulations are intended to reinforce trends in the experimental results and to identify differences between the live experiments and human-in-the-loop simulations. Conducting these runs in simulation allowed for a more consistent starting position for the robot and more consistent initialization timing between the human driven obstacle and the robot. All other

simulation conditions were identical to the conditions in the live experiment runs.

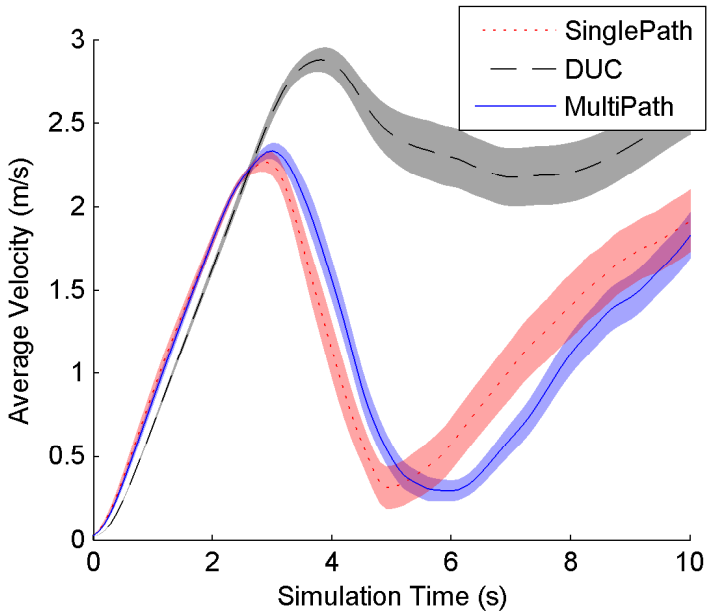


Figure 4.20: Average velocity as a function of simulation time.

Figures 4.21 and 4.20 present the average brake pressure and velocity of the robot over the single human obstacle simulations. During these simulations, the robot is consistently initialized close to the intersection and the robot and the human obstacle driver are initialized at the precisely the same moment. As a result, the robot experiences more consistent obstacle interactions compared to the experimental trials. This causes the robot to experience a higher likelihood of being forced to slow down or stop completely, resulting in a delayed braking, as opposed to the reduced braking observed in Section 4.5.

Figure 4.22 shows the minimum distance between the robot and the human driven obstacle versus the robot's velocity at the minimum distance point. Like with the experiments, the DUC planner results are clustered in the top left cor-

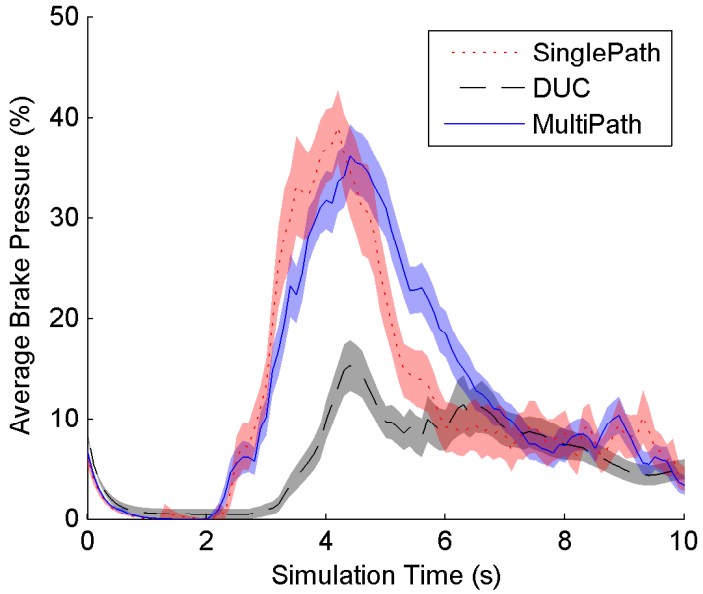


Figure 4.21: Average brake pressure as a function of simulation time.

ner of the plot indicating a significantly higher frequency of close calls and dangerous obstacle interactions compared to the SinglePath and MultiPath planners.

During these simulations, the SinglePath planner experienced no collisions while the MultiPath planner experienced a single not-at-fault collision and the DUC planner experienced four at-fault collisions ($n_{\text{coll}}^{\text{fault}} = 7.1\%$). This increase in collision frequency for the DUC planner suggests that the the safety of the DUC planner is strongly dependent on the aggressiveness and driving style of the human obstacle drivers.

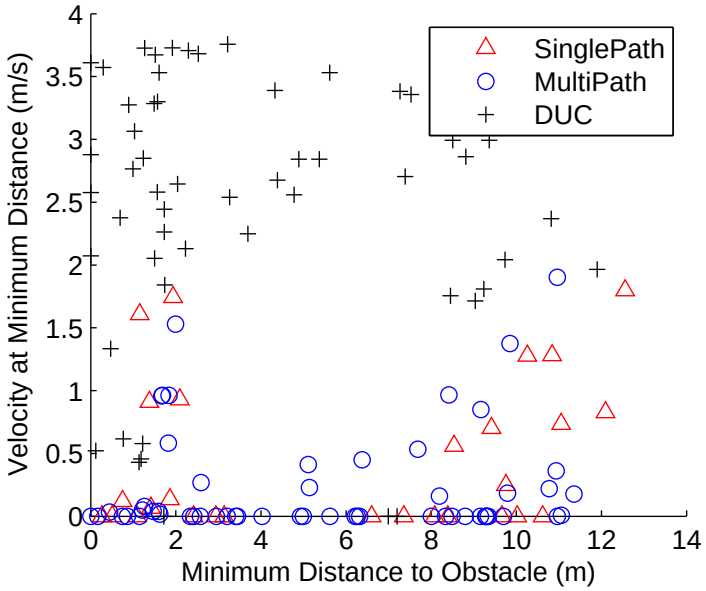


Figure 4.22: Velocity at minimum distance to obstacle for each simulation run.

4.7.2 Multiple Human Obstacle Simulation Results

A set of human-in-the-loop simulations was run using two human driven obstacles to evaluate the safety and performance of the SinglePath and MultiPath algorithms in realistic multiple obstacle interactions. A total of 48 simulation runs were performed for each planner type using two sets of human obstacle drivers. Figure 4.7 shows the experiment setup. Here, the robot is initialized at a fixed location while the virtual obstacle drivers are initialized along one of the three remaining road segments leading into the intersection. This variation in starting point ensures that the robot and the human obstacle drivers experience traffic from all directions. Additionally, to the human obstacle driver, both the robot and the other human driven obstacle are displayed using visually identical models. This ensures that it is difficult for the human obstacle driver to

identify which obstacle is the robot.

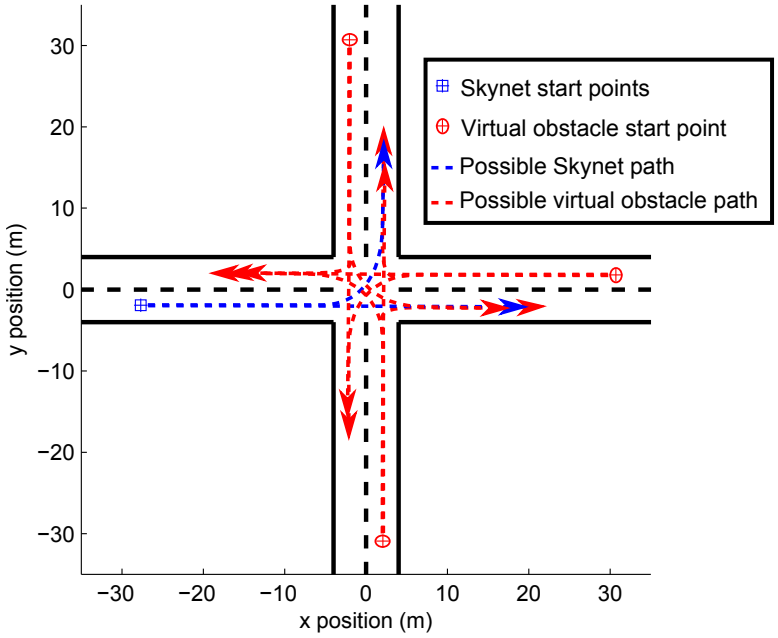


Figure 4.23: Multiple obstacle simulation setup with possible robot maneuvers in blue and virtual obstacle maneuvers in red.

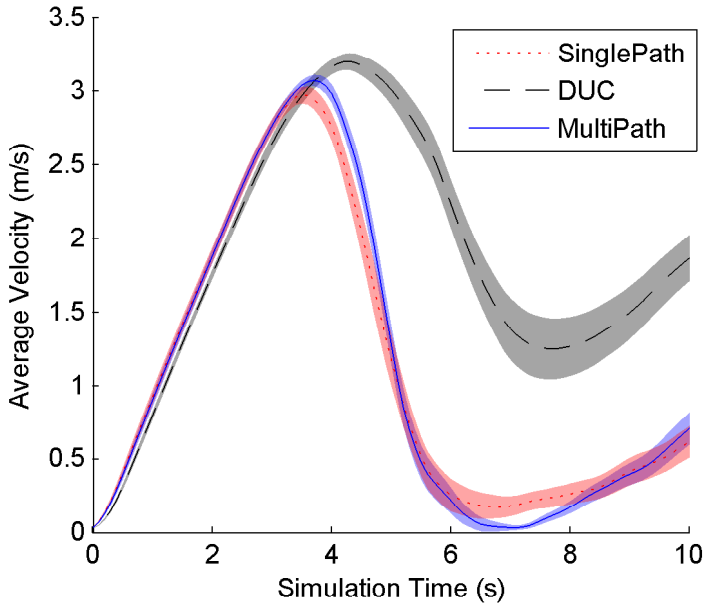


Figure 4.24: Average velocity as a function of simulation time.

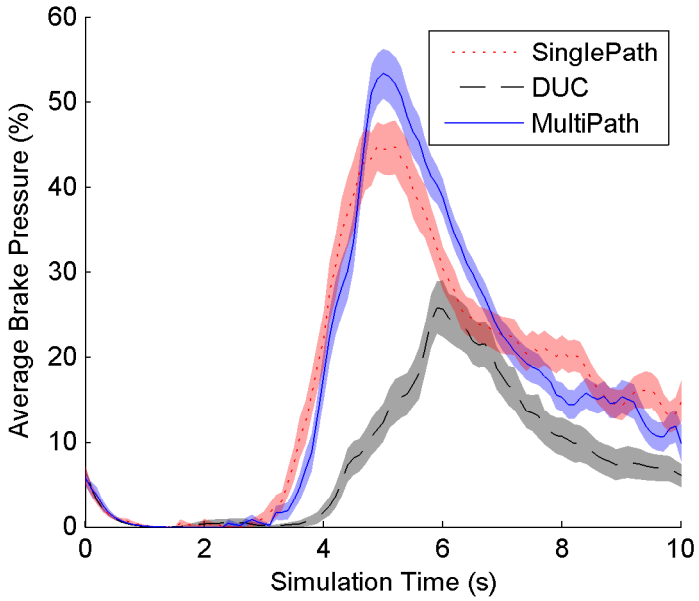


Figure 4.25: Average brake pressure as a function of simulation time.

Figures 4.25 and 4.24 present the average brake pressure and velocity of the robot over the multiple human obstacle simulations. Like with the single obstacle simulations, the MultiPath planner is slightly more aggressive than the SinglePath planner, however, since the robot is almost always forced to yield to the human obstacles, this aggressiveness comes at the cost of an increase in peak brake pressure and a slower recovery from stopping.

Figure 4.26 shows the minimum distance between the robot and the two human driven obstacles vs. the robot's velocity at the minimum distance point. Like in the single obstacle simulations, the DUC planner experiences the greatest number of dangerous obstacle interactions, while the SinglePath and MultiPath planners are significantly more cautious.

Table 4.1 presents the at-fault and not-at-fault collision frequencies for the multiple human obstacle simulations. The virtual obstacle collision frequencies

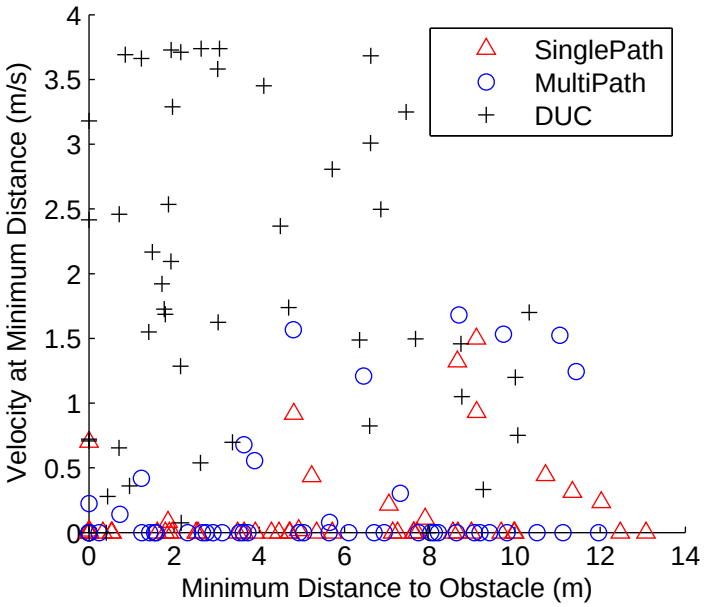


Figure 4.26: Velocity at minimum distance to obstacles for each simulation run.

Table 4.1: Collision Frequencies

	Not-at-fault	At-fault
SinglePath Collisions	14.6%	0%
MultiPath Collisions	12.5%	0%
DUC Collisions	2.1%	14.6%
Virtual Obstacle Collisions	2.8%	7.0%

indicate the number of collisions that occurred between the two virtual obstacle drivers as they attempted to navigate the uncontrolled intersection. Virtual obstacle collisions are considered at-fault when both vehicles are partially responsible. Not-at-fault virtual obstacle collisions occur when one human driven obstacle yields safely and is then hit by the other driver due to oversteer or un-

dersteer.

Both the SinglePath and the MultiPath planners experienced a large number of not-at-fault collisions during the multiple human obstacle simulations. This increase in collision frequency is due to the obstacle drivers having the additional distraction of another vehicle to avoid. However, both planners still experience zero at-fault collisions. The DUC planner experienced a substantial increase in at-fault collision probability indicating that reactive planning is not sufficient to handle complex collision avoidance scenarios with multiple interacting obstacles. The human drivers also struggled with this scenario, exhibiting a significant number of at-fault collisions where both drivers made poor collision avoidance decisions. This highlights the difficulty of safely navigating the tested intersection scenario.

4.8 Conclusions

A series of human-in-the-loop experiments and simulations are presented to investigate the performance benefits of two key components of the motion planning and collision avoidance layer of Cornell's autonomous vehicle Skynet: 1) the inclusion of a discrete set of short term continuous obstacle motion predictions for each dynamic obstacle in the environment, and 2) planning multiple contingency paths to account for the mutually exclusive nature of these obstacle predictions. A virtual obstacle interface was also developed to enable the safe testing of dangerous collision avoidance scenarios while providing realistic human driven obstacle behavior.

Testing demonstrated that the MultiPath planner is more aggressive in terms

of average velocity and reduced or delayed braking than the SinglePath planner with no associated increase in at-fault collision frequency. Simulation results using prerecorded human driven obstacle data demonstrate that this performance advantage persists over a wide range of obstacle interaction timings. Additionally, the contingency planner is capable of planning up to four simultaneous contingency paths in scenarios with three dynamic obstacles with 95% confidence that the path optimization will terminate in under 0.5 seconds.

The SinglePath and MultiPath planners both utilized dynamic obstacle motion predictions and exhibited significant safety and collision avoidance advantages over the reactive DUC planner. Neither planning approach experienced an at-fault collision during human-in-the-loop testing. In tests with two simultaneous human driven obstacle vehicles, the SinglePath and MultiPath planner were not only safer than the DUC planner, they also experience significantly fewer at-fault collisions than the human obstacle drivers.

These results indicate that utilizing even low-fidelity predictions of future obstacle motion provides a substantial improvement in collision avoidance capabilities for autonomous driving systems. Additionally, planning multiple contingency paths allows a robot to plan more aggressive actions when using mutually exclusive sets of obstacle motion predictions. These results also demonstrate that contingency planning is capable of real-time performance even during complex multiple obstacle scenarios.

Acknowledgments

This research is partially supported by ARO Grant #W911NF-09-1-0466, with Dr. Randy Zachery as Program Manager

CHAPTER 5

MULTIPLE-STEP PREDICTION USING ADAPTIVE GAUSSIAN MIXTURES FOR A TWO STAGE GAUSSIAN PROCESS MODEL

Abstract

A two stage probabilistic prediction model is presented that uses nonparametric Gaussian Process (GP) regression to model continuous complex actions combined with a parametric model for known system dynamics. This two stage model is applied to the case of anticipating driver behavior and vehicle motion. Prediction over the GP driver behavior model is performed by analytically evaluating the expectation integrals for the moments of the output distribution. An on-the-fly data selection technique is proposed to mitigate computational scaling concerns when analytically computing higher order moments and an adaptive Gaussian mixture model (GMM) formulation is presented to handle nonlinear and multimodal prediction problems. This adaptive GMM formulation includes a method for analytically evaluating the non-Gaussianity of the output distribution and adaptively splitting the initial state distribution into a sum of Gaussians to reduce the effect of nonlinearities on prediction accuracy. The proposed prediction model and adaptive GMM formulation are evaluated using driving data collected from three human subjects navigating a standard four-way intersection in a driving simulation.

5.1 Introduction

Systems comprised of known physical dynamics and highly uncertain decision making processes are difficult to model using only parametric or only nonparametric model techniques. Parametric models are well-suited for modeling well-understood physical systems, but have difficulty capturing complex or poorly understood behavior. Nonparametric models such as Gaussian Process (GP) regression models are well suited for modeling complex, highly uncertain systems such as intelligent agents, advanced controllers, or poorly understood physical phenomena. However, nonparametric models are not well suited for modeling systems with well understood dynamics and important physical constraints, such as a nonholonomic vehicle.

A common approach when using GP regression to model complex systems with known underlying dynamics is to include the dynamics in the mean function used by the GP [82]. While this approach allows a GP regression model to track the underlying dynamics of a system, it does not allow for the inclusion of parametric constraints on the system dynamics. Including system dynamics through the mean function is also a poor approximation for hierarchical systems in which there exists a clean separation between the states modeled by the GP and the states used in the known dynamics model.

A motivating example used in this paper is anticipating the motion of a dynamic obstacle with an intelligent controller such as a road vehicle. Anticipating dynamic obstacle motion is a key enabling step in planning and collision avoidance systems for mobile robots in dynamic environments. Producing accurate motion predictions for complex goal-oriented dynamic obstacles such as road

vehicles requires modeling obstacle intentions as well as the physical dynamics of the obstacle's motion. Obstacle predictions based on inferred obstacle goals have been applied to mobile robots [63], autonomous road vehicles [17] [3], and air traffic control systems [37].

GP regression has been used to model the motion of road vehicles. Kim et al. [44] and Joseph et al. [40] both use GP regression to model the continuous state derivatives of a vehicle's motion given the current vehicle state. Laugier et al. [51] sample from a GP regression model of observed trajectories to perform collision risk assessment for obstacle vehicles.

A two stage prediction model is proposed here consisting of a nonparametric GP regression model, coupled with a parametric system dynamics model. A key challenge in performing multiple-step prediction using GP regression models is evaluating the GP over a distribution of input states. Ko and Fox [45] show that a GP dynamics model can be used in standard Bayesian filter formulations including an Extended Kalman Filter (EKF), an Unscented Kalman Filter (UKF), and a particle filter. However, the GP Bayesian filtering methods presented in [45] approximate the propagated distribution by treating the GP as a point-to-point mapping, while evaluating the output uncertainty only at the mean of the input distribution. This can be an effective propagation strategy for sampling based filtering techniques such as the UKF and particle filter, but sampling over a GP is computationally challenging. Candela et al. [11] show that these approximations can be avoided for a certain class of kernel functions by analytically evaluating the expectation integrals of the output distribution.

The analytical propagation proposed by [11] works well for purely nonparametric GP models. Quinonero-Candela et al. [69] also derive a closed form so-

lution for the cross covariance between the GP input and output distributions for a general class of GPs. This cross covariance information is critical in the proposed two stage system model because it allows the proposed model to capture state dependent control behaviors such as lane keeping tendencies for road vehicles. A key drawback to analytically computing the moment integrals of the output distribution is poor computational scaling for higher order moments such as the output variance. An on-the-fly data selection procedure is proposed that allows higher order moments to be computed using only a small subset of the most relevant data from the GP models. This procedure is independent of, and can be applied in addition to, global sparse GP strategies, such as [28], which seek to reduce the entire GP data set to a smaller representative data set.

The rest of the paper is organized as follows. Section 5.2 outlines the proposed two stage system model. Section 5.3 provides a brief overview of GP regression models. Section 5.4 details an analytical solution for the GP output evaluated over an input distribution as well as for the cross covariance terms between the GP output and input distributions. Section 5.4 also presents a novel method for on-the-fly compression of the GP data set to improve computational efficiency. Section 5.5 presents a prediction algorithm for the proposed two stage model. Section 5.6 presents an adaptive Gaussian mixture model prediction method for nonlinear and multimodal systems. Section 5.7 presents a two stage driver-vehicle model for predicting the motion of a road vehicle. Section 5.8 provides experimental results based on data collected from human subjects in a driving simulation. Section 5.9 provides conclusions.

5.2 Two Stage System Model

Figure 5.1 shows the proposed two stage system model. In the first stage, GP regression is used to model the affects of complex, highly uncertain aspects of the system. In the second stage a parametric model is used to model the portion of the system with well understood dynamics.

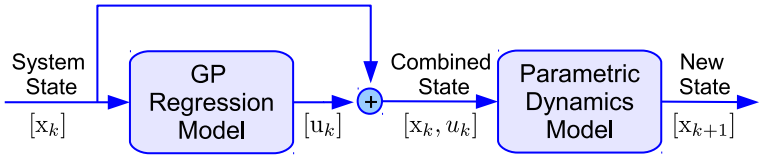


Figure 5.1: Block diagram of the two stage system model consisting of a GP regression model combined with a parametric system dynamics model.

This is a hierarchical model where the first stage only depends on the current state of the system, while the second stage takes as input both the system state and the control output of the first stage. For the case of a dynamic obstacle such as a road vehicle, the first stage represents an intelligent controller which produces control actions based on the current state of the system and the second stage represents the physical dynamics of the obstacle, which depend on both the current system state and the control actions from the intelligent controller.

5.3 Gaussian Process Overview

A GP regression model is a probabilistic regression model that represents a distribution over functional mappings from an input space to an output space where all the outputs are jointly distributed as Gaussian. For a set of train-

ing data $\mathcal{D} = \{\mathcal{D}_x, \mathcal{D}_y\}$, where $\mathcal{D}_x = \{x_i\}_{i=1}^N$ is the set of input data points and $\mathcal{D}_y = \{y_i\}_{i=1}^N$ is the set of output data points, a GP regression model provides a distribution over the function $y_i = g(x_i) + \epsilon_i$, where $\epsilon_i \sim \mathcal{N}(0, \sigma_n^2)$. A GP, given its data, is fully specified by its mean, $M(x)$, and covariance, $C(x_i, x_j)$, functions. For this paper, the standard zero mean function is used, $M(x) = 0$, and the covariance function is an autoregressive kernel function using the popular squared exponential form:

$$C(x_i, x_j) = \sigma_f^2 \exp\left(-\frac{1}{2}(x_i - x_j)^T \Lambda^{-1}(x_i - x_j)\right) + \sigma_n^2 \delta(x_i, x_j) \tag{5.1}$$

where Λ is a diagonal weighting matrix, and σ_f^2, Λ, and σ_n^2 are covariance function parameters that are fit by maximizing the marginal likelihood of the hyper parameters given the training data [70].

For a GP regression model with known parameters, the predictive distribution at an arbitrary point in the input space, x^*, is characterized by its mean and variance:

$$\begin{aligned} p(g(x^*)|x^*) \sim & \mathcal{N}\left(\mu_y(x^*), \sigma_y^2(x^*)\right) \\ \mu_y(x^*) = & \mathbf{k}^{*T}\mathbf{K}^{-1}\mathbf{y}, \quad \sigma_y^2(x^*) = \mathbf{k}^{**} - \mathbf{k}^{*T}\mathbf{K}^{-1}\mathbf{k}^* \end{aligned} \tag{5.2}$$

where $\mathbf{k}^* = [C(x^*, x_1), \ldots, C(x^*, x_N)]^T$ represents the covariance function evaluated between the input point and each data point, $\mathbf{K}_{ij} = C(x_i, x_j)$ represents the covariance function evaluated for each pair of data points, and $\mathbf{k}^{**} = C(x^*, x^*)$ is the covariance function evaluated between the input point and itself.

5.4 Analytical Evaluation of GPs at Uncertain Inputs

Multi-step prediction of the proposed two stage system model requires evaluating the GP over an input distribution instead of an input point. Candela et al. [11] show that the first and second moments of the GP output given a normally distributed input have exact, closed-form solutions for Gaussian kernel functions, including the squared exponential kernel function given in Equation 5.1. Accurate prediction using the proposed two stage model also requires computing the cross covariance between the input and output of the GP in order to define the full combined state distribution. This combined state distribution is then used as an input to the second stage parametric dynamics model. Subsections 5.4.1 and 5.4.2 provide an overview of the analytical moment evaluations proposed by Candela et al. [11] for mean and variance and for input-output cross covariance respectfully. Subsection 5.4.3 presents an on-the-fly data selection technique for greatly reducing the computational requirements of evaluating these analytical solutions.

5.4.1 GP Mean and Variance at Uncertain Input

If the GP input x^* is normally distributed such that $x^* \sim \mathcal{N}(\mu_x, \Sigma_x)$, the GP evaluation in Equation 5.2 becomes instead:

$$p\left(g(x^*)|\mu_x, \Sigma_x\right) = \int p\left(g(x^*)|x^*\right) \mathcal{N}(x^*|\mu_x, \Sigma_x) dx^* \tag{5.3}$$

This integral is intractable and can not be solved in closed form. However, the mean and variance can be exactly calculated in closed-form, and the GP output distribution can be approximated by a normal distribution with the same mean

and variance as the true distribution:

$$
\begin{aligned}
p\left(g(x^*)|\mu_x, \Sigma_x\right) &\approx \mathcal{N}\left(\mathrm{m}(\mu_x, \Sigma_x), \mathrm{var}(\mu_x, \Sigma_x)\right) \\
\mathrm{m}(\mu_x, \Sigma_x) &= E_{x^*}\left[E_{g(x^*)}\left[g(x^*)|x^*\right]\right] \\
&= E_{x^*}\left[\mu_y(x^*)\right] \\
\mathrm{var}(\mu_x, \Sigma_x) &= E_{x^*}\left[\mathrm{Var}_{g(x^*)}(g(x^*)|x^*)\right] + \\
&\quad \mathrm{var}_{x^*}(E_{g(x^*)}\left[g(x^*)|x^*\right]) \\
&= E_{x^*}\left[\sigma_y^2(x^*)\right] + \mathrm{var}_{x^*}(\mu_y(x^*))
\end{aligned} \tag{5.4}
$$

Quinonero-Candela et. al. provide exact, closed form expressions for the mean and variance [69].

5.4.2 GP Input-Output Covariance

In order to correctly propagate a state distribution through a two-stage model, the full joint distribution of the inputs to the second stage must be available. The full joint distribution is:

$$
\begin{bmatrix} x^* \\ g(x^*) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_x \\ \mathrm{m}(\mu_x, \Sigma_x) \end{bmatrix}, \begin{bmatrix} \Sigma_x & \Sigma_{xy} \\ \Sigma_{xy}^T & \mathrm{var}(\mu_x, \Sigma_x) \end{bmatrix}\right) \tag{5.5}
$$

and the GP input-output covariance is defined as:

$$
\begin{aligned}
\Sigma_{xy} &= \mathrm{Cov}\left(g(x^*), x^*\right) \\
&= E_{x^*}\left[E_{g(x^*)}\left[g(x^*)|x^*\right] x^*\right] - E_{x^*}\left[E_{g(x^*)}\left[g(x^*)|x^*\right]\right] E_{x^*}\left[x^*\right]
\end{aligned} \tag{5.6}
$$

As with the mean and variance (Equation 5.4), a closed-form expression for the input-output correlation is available in [69]. Ignoring the GP input-output correlation is similar to assuming that the output of the first stage of the model is

independant of the original state distribution while evaluating the second stage. While this is a comman assumption when using GPs in a predictive framework, results in Section 5.8 will show that it is a poor approximation when using a two-stage model.

5.4.3 On-the-Fly Data Selection

Sections 5.4.1 and 5.4.2 describe an exact method for finding the mean, variance, and input-output cross covariance for a GP evaluated over a normally distributed input. Assuming that $\mathbf{K}^{-1}$ is precomputed before performing any prediction, the computational complexity of the analytical calculation of the output variance is $\mathcal{O}(N^2)$, where N is the number of data points in the GP. This scaling with the square of the size of the data set means that evaluating the analytical solution can be much more computationally expensive than numerical methods such as the sigma point transform used in the UKF.

An approximation is proposed here that uses only a small subset of the most relevant GP data for each higher order moment evaluation. The mean $\mathrm{m}(\mu_x, \Sigma_x)$ in Equation 5.4 is given by [69] as:

$$\mathrm{m}(\mu_x, \Sigma_x) = \mathbf{y}^T \mathbf{K}^{-1} \mathbf{l} \tag{5.7}$$

where $\mathbf{l} = [l_1, ..., l_N]^T$ is given by:

$$l_i = \int C(x^*, x_i)\, \mathcal{N}(x^* | \mu_x, \Sigma_x) dx^* \tag{5.8}$$

The vector $\mathbf{l} = [l_1, ..., l_N]^T$ is similar to $\mathbf{k}^*$ in the standard GP evaluation in Equation 5.2. Both terms are vectors that measure the relevance each data point has to the input distribution or input point respectfully. In many systems, only a

small fraction of the GP data is relevant to a given input. This means that the output variance can be well approximated using only the n most relevant data points, where $n << N$. This data selection step requires inverting the reduced data covariance matrix, $\mathbf{K}_{\text{red}}^{-1}$, which is $\mathcal{O}(n^3)$. This yields variance calculations that are $\mathcal{O}(N+n^3)$ instead of $\mathcal{O}(N^2)$. In Section 5.8, it is demonstrated that using this data selection method provides significant computational advantages over numerical methods while still closely approximating the true solution.

5.5 Two Stage GP Prediction Algorithm

The proposed prediction algorithm uses the results in Equation 5.4 and Equation 5.6 to perform a two-part propagation of the state distribution through the model described in Figure 5.5. As a preprocessing step, a GP is learned offline for each of the n_u elements of u_k, denoted $\{\text{GP}_1, ..., \text{GP}_{n_u}\}$. During the first prediction stage, each of the learned GP models is evaluated over the initial system state distribution $x_k \sim \mathcal{N}(\mu_k, \Sigma_k)$, resulting in distributions over the GP output ($u_j \sim \mathcal{N}(\mu_j, \Sigma_j)$), as well as the cross covariance terms between GP outputs and the input state (Σ_{xj}). In the second prediction stage the system state is propagated through the parametric dynamics model using the combined distribution of the system state and the GP outputs. This propagation is performed using a sigma point transform [41]. This propagation algorithm applied to the system shown in Figure 5.1 is outlined in Algorithm 1.

Algorithm 1 has two distinct advantages over existing GP propagation algorithms. First, the composition of GP regression models with a parametric dynamics model makes it possible to capture both complex or poorly understood

Algorithm 1: The two stage GP propagation algorithm

Require: $\mathbf{x}_k \sim \mathcal{N}(\mu_k, \Sigma_k)$
Evaluate the GPs:
for $j = 1 \rightarrow n_u$ **do**
$\left[\mu_j, \sigma_j^2, \Sigma_{xj}\right] \leftarrow \mathrm{GP}_j(\mu_k, \Sigma_k)$
end for
$\boldsymbol{\mu}_u \leftarrow [\mu_1, ..., \mu_{n_u}]^T$
$\Sigma_u \leftarrow \mathrm{diag}(\left[\sigma_1^2, ..., \sigma_{n_u}^2\right])$
$\Sigma_{xu} \leftarrow [\Sigma_{x1}, ..., \Sigma_{xn_u}]$
Perform the sigma point propagation:
Find $\mathbf{S}_x = [\mathbf{S}_x^1, ..., \mathbf{S}_x^{n_x}]$ such that $\mathbf{S}_x \cdot \mathbf{S}_x^T = \Sigma_x$
Find $\mathbf{S}_u = [\mathbf{S}_u^1, ..., \mathbf{S}_u^{n_u}]$ such that $\mathbf{S}_u \cdot \mathbf{S}_u^T = \Sigma_u$
Given sigma point spread parameter λ
Given sigma point weights $[w_m^1, ..., w_m^{2n_x+2n_u}]$ *and* $[w_c^0, ..., w_c^{2n_x+2n_u}]$
for $i = 0 \rightarrow 2n_x + 2n_u$ **do**
if $i \in [0, 2n_x + 1, ..., 2n_x + 2n_u]$ **then**
$\chi_k^i \leftarrow \mu_x$
else if $i \in [1, ..., n_x]$ **then**
$\chi_k^i \leftarrow \mu_x + \lambda \cdot \mathbf{S}_x^i$
else
$\chi_k^i \leftarrow \mu_x - \lambda \cdot \mathbf{S}_x^{i-n_x}$
end if
if $i \in [0, ..., 2n_x]$ **then**
$\nu_k^i \leftarrow \boldsymbol{\mu}_u + \Sigma_{xu}^T \cdot (\chi_k^i - \mu_k)$
else if $i \in [2n_x + 1, ..., 2n_x + n_u]$ **then**
$\nu_k^i \leftarrow \boldsymbol{\mu}_u + \lambda \cdot \mathbf{S}_u^{i-2n_x}$
else
$\nu_k^i \leftarrow \boldsymbol{\mu}_u - \lambda \cdot \mathbf{S}_u^{i-2n_x-n_u}$
end if
Evaluate the parametric model
$\bar{\chi}_{k+1}^i \leftarrow f(\chi_k^i, \nu_k^i)$
end for
Find first two moments of propagated state
$\mu_{k+1} \leftarrow \sum_{i=0}^{2n_x+2n_u} w_m^i \cdot \bar{\chi}_{k+1}^i$
$\Sigma_{k+1} \leftarrow \sum_{i=0}^{2n_x+2n_u} w_c^i \cdot \left(\bar{\chi}_{k+1}^i - \mu_{k+1}\right) \cdot \left(\bar{\chi}_{k+1}^i - \mu_{k+1}\right)^T$
$p(\mathbf{x}_{k+1}) \approx \mathcal{N}(\mathbf{x}_{k+1} | \mu_{k+1}, \Sigma_{k+1})$

behavior using the GP and to capture and enforce well-understood underlying physical processes using the parametric model. Second, only a single GP evaluation is required per propagation step due to the use of closed form analytical expressions for the output distribution and cross covariance terms.

5.6 Adaptive Gaussian Mixture Formulation

Approximating the output distribution of a Gaussian Process evaluated over an input distribution using the first and second moment calculations presented in Section 5.4 can account for some degree of nonlinearity as long as the output distribution is well modeled by a Gaussian distribution. When the output distribution is not well modeled by a Gaussian distribution, this approach produces a poor approximation of the true output distribution. Non-Gaussianity in the output distribution can arise due to complex nonlinearities in the GP data, such as multi-modal system behaviors.

If the nonlinearities in the GP mapping are smooth, the true output distribution can be better approximated by decomposing the input distribution into a Gaussian Mixture Model (GMM), where each mixture element is smaller in terms of variance than the original input distribution. This adaptive GMM approach has been applied to general nonlinear estimation and prediction problems [68],[36],[74]. Here it is adapted to the problem of estimating the output of a GP regression model evaluated over an input distribution.

The two necessary elements for performing adaptive splitting are 1) a method for detecting when splitting is necessary, and 2) a vector along which to split the input distribution. Once the need for a split is detected, and a splitting

direction is known, the input distribution can be normalized and a precomputed GMM approximation can be applied as in [36]. Section 5.6.1 presents an analytical kurtosis based metric for detecting non-Gaussianity in the GP output. Section 5.6.2 presents a one dimensional example of adaptive GMM propagation for a GP model of a nonlinear function with a bimodal output.

5.6.1 Analytical Kurtosis Evaluation for Non-Gaussianity Detection

The kurtosis of a distribution is an established metric of non-Gaussianity [38]. The excess kurtosis of a distribution is defined as $\mathrm{kurt}(y) = \frac{E[y^4]}{(E[y^2])^2} - 3$, where the excess kurtosis for a Gaussian distribution is zero. Typically, other metrics of non-Gaussianity such as Negentropy are preferable due to concerns about sensitivity to outliers when computing the kurtosis based on a set of samples. Here, the excess kurtosis of the output distribution can be directly computed by analytically solving the required expectation integrals.

Using excess kurtosis as a measure of non-Gaussianity, splitting is performed whenever the magnitude of the excess kurtosis of the GP output surpasses a predefined threshold. The Appendix provides a derivation for analytically calculating the excess kurtosis of the GP output distribution. An important note about the kurtosis calculation is that its computational complexity scales as $\mathcal{O}(N^4)$. This scaling prevents computationally efficient application of the kurtosis metric for GP models with more than a small number of data points. The on-the-fly data selection method presented in Section 5.4.3 allows the kurtosis metric to be computed efficiently using only a small subest of the most relevant

training data points. The eigenvector associated with the largest eigenvalue of the input covariance is chosen as the splitting direction as in [74].

5.6.2 Adaptive GMM Example - Univariate Nonstationary Growth Model

The univariate nonstationary growth model (UNGM) is a highly nonlinear function that is commonly used as a benchmark nonlinear estimation function. The UNGM is defined as:

$$y = f(x) = \alpha x + \beta \frac{x}{1 + x^2} + \gamma \cos(1.2(t-1)) \tag{5.9}$$

For this paper, the following parameter values are used: $\alpha = 1.5, \beta = 15, \gamma = 8, t = 1$. The UNGM is typically used as a time-series model; here it is used as a simple nonlinear function.

Figure 5.2 shows the UNGM with noise, $y = f(x) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$, which maps a Gaussian distribution over the input space into a bimodal distribution in the output space. In Figure 5.3, samples from this noisy UNGM model are used to train a GP regression model and the GP output is approximated using the analytical mean and variance as presented in Section 5.4.

The adaptive splitting approach outlined in this section is then applied using a kurtosis threshold of $k_{\max} = 0.3$. When an unacceptable level of non-Gaussianity is detected, the input distribution is split into a mixture of 3 component distributions as in [36]. This process is repeated recursively until all output distributions satisfy the kurtosis threshold or until a predefined recursion depth is reached. The recursion depth for this example is $n_{\max}^{\text{rec}} = 3$.

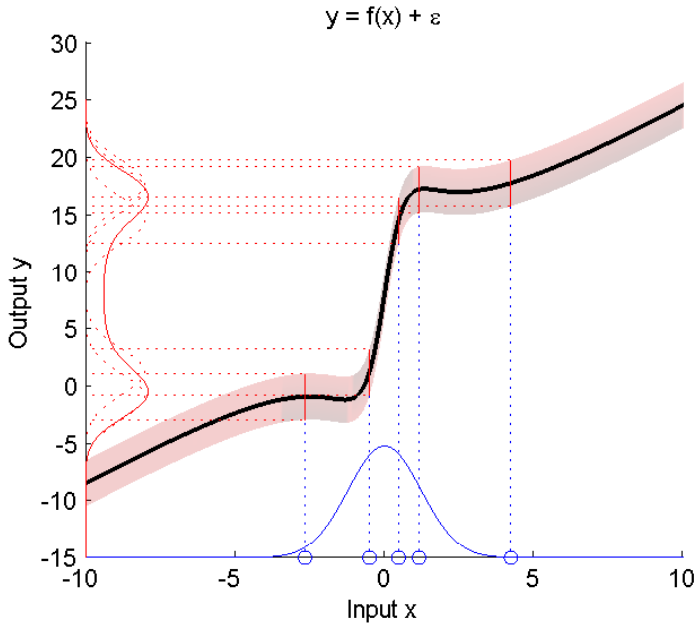


Figure 5.2: A Gaussian input mapped through the UNGM with noise.

The prediction using adaptive splitting shown in Figure 5.4 produces a much better estimate of the true output distribution. The Kullback-Leibler divergence (KLD) between the predicted distribution and the true output distribution is $KLD_{\text{split}} = 0.033$ for the adaptive splitting approach and $KLD_{\text{no split}} = 0.319$ for the non-splitting approach.

5.7 Driver-Vehicle Model

The two stage prediction model presented in Section 5.2 is applied to the case of predicting the driving behavior and vehicle motion of a road vehicle. The first stage consists of driver decision making; the driver takes as inputs information about the environment, the state of the vehicle, and internal driving goals and outputs a set of vehicle commands controlling the steering and forward acceler-

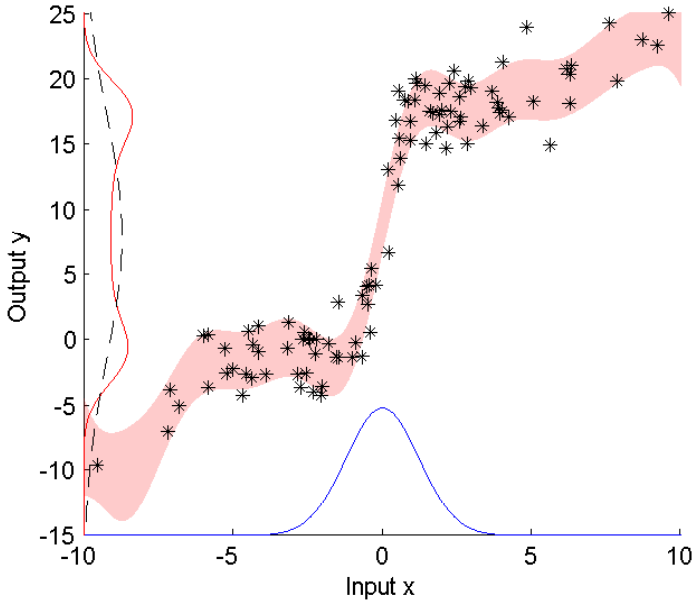


Figure 5.3: Analytical propagation with no splitting through a GP model trained from a noisy UNGM model. The true output distribution is shown in red. The estimated output distribution, using the analytically computed mean and variance, is shown in black.

ation of the vehicle. This stage represents a complex, highly uncertain decision making process and is not well modeled using parametric modeling techniques. The second stage consists of the vehicle dynamics wherein the driver commands are applied to the vehicle and the vehicle state is propagated forward in time. This stage represents a known physical process and can be well modeled using a range of parametric models.

Figure 5.5 shows a two stage driver-vehicle model consisting of a nonparametric GP driver behavior regression model, followed by a parametric vehicle dynamics model. The vehicle state description used in this model is a four dimensional state vector which includes the two dimensional position of the ve-

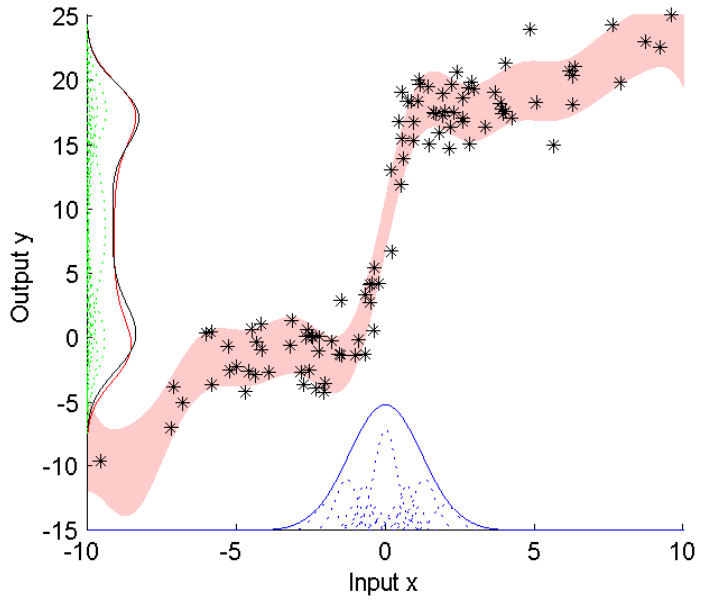


Figure 5.4: Analytical propagation with splitting through a GP model trained from a noisy UNGM model. The true output distribution is shown in red. The estimated output distribution, using the adaptive splitting approach with a kurtosis threshold of $k_{\max} = 0.3$, is shown in black.

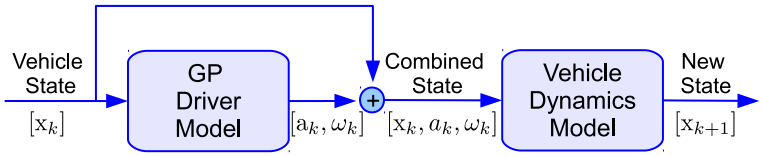


Figure 5.5: Block diagram of the two stage driver-vehicle model consisting of a GP driver behavior model combined with a parametric vehicle dynamics model.

hicle along with its two dimensional velocity vector:

$$\mathbf{x}_k = \begin{bmatrix} x & y & v_x & v_y \end{bmatrix}^T \tag{5.10}$$

At the current timestep k, the vehicle state $\mathbf{x}_k$ is assumed to be normally distributed with a known mean and covariance, $\mathbf{x}_k \sim \mathcal{N}(\mu_{\mathbf{x}}, \Sigma_{\mathbf{x}})_k$. The driver behavior model consists of two independently trained GP regression models

for throttle commands expressed as forward acceleration, a_k, and steering commands expressed as turn rate, ω_k:

$$a_k \sim \mathrm{GP_a}(\mathbf{x}_k, \mathcal{D}_x, \mathcal{D}_a) \tag{5.11}$$

$$\omega_k \sim \mathrm{GP}_\omega(\mathbf{x}_k, \mathcal{D}_x, \mathcal{D}_\omega) \tag{5.12}$$

where $\mathcal{D}_x$ is the set of vehicle states used as input data in the GP regression model, and $\mathcal{D}_a$ and $\mathcal{D}_\omega$ are the set of output data points corresponding to forward acceleration commands and steering commands respectfully.

These driving commands are then used as control inputs in the vehicle dynamics model. Parametric models for the dynamics of road vehicles are well studied; for this paper, the vehicle dynamics are represented by a constant curvature, constant acceleration bicycle model. Schubert et al. [73] show that despite its simplicity, this model is as effective as more sophisticated vehicle dynamics models for predicting vehicle motion. The vehicle dynamics model is defined in Equation 5.13.

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, a_k, \omega_k, \delta t) \tag{5.13}$$

This equation can be rewritten with the function $f(\mathbf{x}_k, a_k, \omega_k, \delta t)$ explicitly defined as:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ v_{x,k+1} \\ v_{y,k+1} \end{bmatrix} = \begin{bmatrix} x_k + \Delta x \cos(\theta_k) - \Delta y \sin(\theta_k) \\ y_k + \Delta y \sin(\theta_k) + \Delta x \cos(\theta_k) \\ (v_k + a\delta t)\cos(\theta + \Delta\theta) \\ (v_k + a\delta t)\sin(\theta + \Delta\theta) \end{bmatrix} \tag{5.14}$$

where

$$\begin{aligned}
\rho_k =& \frac{v_k}{\omega_k} \\
l_k =& \frac{1}{2} a_k \delta t^2 + v_k \delta t \\
\Delta\theta =& l_k \frac{1}{\rho_k} \\
\Delta x =& \rho_k \sin(\Delta\theta) \\
\Delta y =& \rho_k (1 - cos(\Delta\theta)) \\
v_k =& \sqrt{v_{x,k}^2 + v_{y,k}^2}
\end{aligned}$$

5.8 Experimental Evaluation

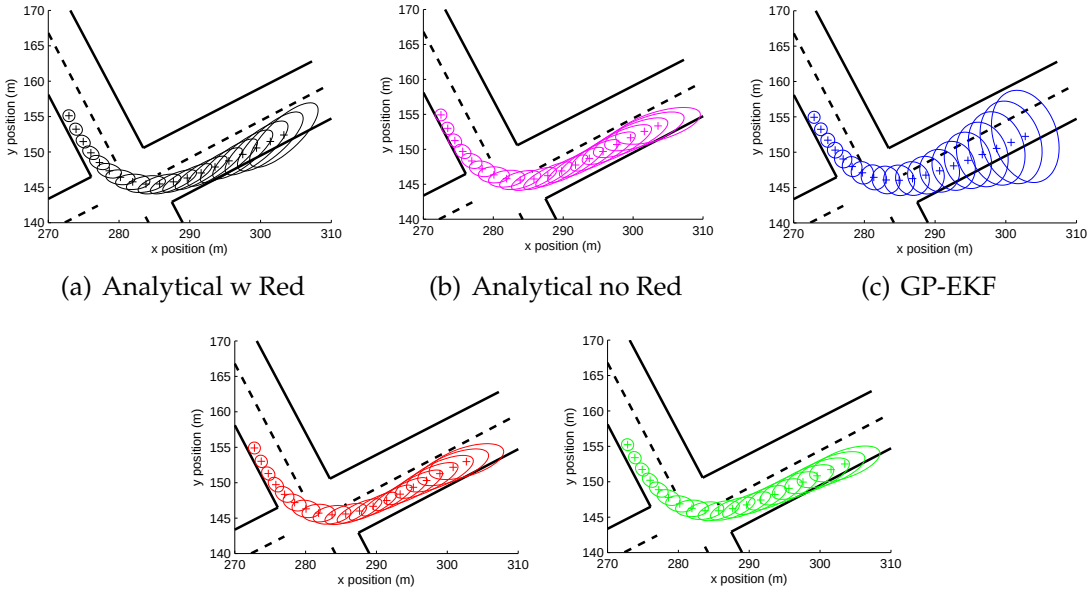
A four way road intersection scenario is used to test the analytical GP propagation presented in Algorithm 1 using the two stage driver-vehicle model presented in Section 5.7. Driving behavior data from three human test subjects was collected using the open source driving simulation TORCS [15]. Each test subject was asked to perform 10 runs each for four distinct intersection traversal maneuvers: turn left, turn right, drive straight, and stop-at-line. Position and velocity data was recorded for each maneuver at a rate of 5 Hz. Acceleration and steering commands were computed using the inverse of the constant curvature constant acceleration vehicle dynamics model specified in Equation 5.13.

Section 5.8.1 presents prediction results for the turn left, turn right, and drive straight maneuvers using the analytical propagation method outlined in Algorithm 1 and the GP Bayesian propagation strategies proposed in [45]. Section 5.8.2 presents computational scaling results for the on-the-fly data selection procedure outlined in Section 5.4.3. Section 5.8.3 presents prediction results for

multimodal combined maneuvers using the adaptive GMM approach presented in Section 5.6 over a range of kurtosis splitting threshold values.

5.8.1 Prediction of Unimodal Maneuvers

Open loop prediction was performed over a prediction horizon of 20 timesteps to evaluate the propagation accuracy of Algorithm 1 for the turn left, turn right, and drive straight maneuvers. Prediction trials were conducted using a leave-one-out cross validation approach, where the initial state estimate of the left out trajectory is used as the initial state distribution for prediction. The results in this section compare the propagation performance of four algorithms: 1) the analytical GP evaluation defined in Algorithm 1 using the on-the-fly data selection step outlined in Section 5.4.3 with a reduced data size of $n = 30$; 2) the analytical GP evaluation defined in Algorithm 1 with no data reduction; 3) a variation of the propagation procedure defined in Algorithm 1, where the GP evaluations are performed using the prediction step of the GP-EKF presented in [45]; 4) a sigma point prediction update, similar to the GP-UKF presented in [45], applied to the entire two stage driver-vehicle model, where the GP is treated as a point to point mapping and the variance of the GP outputs is evaluated only at the mean of the input distribution. An open loop Monte Carlo prediction was also run using 1000 samples to provide a close approximation of the true predicted distribution given the GP models. The performance of each algorithm is evaluated by computing the KLD between the predicted distribution at each step and the normal distribution given by the first two moments of the Monte Carlo result.



(a) Analytical w Red (b) Analytical no Red (c) GP-EKF

(d) Sigma Point (e) Monte Carlo

Figure 5.6: Left turn prediction results for each algorithm over 20 timesteps. Ellipses represent the prediction error uncertainty at the 50% confidence interval.

Figure 5.6 shows typical predicted vehicle trajectory distributions for each of the tested algorithms using GP models trained on the left turn maneuver data. The prediction results for the analytical GP evaluation with data selection shown in Figure 5.6(a) are qualitatively very similar to the results for the analytical GP evaluation without compression shown in Figure 5.6(b) as well as the results for the sigma point approach shown in Figure 5.6(d) and the Monte Carlo solution shown in Figure 5.6(e). These results suggests that the analytical approach with cross covariance information and the sigma point approach are both capable of capturing lane keeping behavior and are both good approximations of the Monte Carlo result. These results also suggest that the on-the-fly data selection step introduces no significant approximation errors despite the propagation algorithm using an order of magnitude less data. Figure 5.6(c)

shows the predicted vehicle trajectory using the GP-EKF. The prediction results for the GP-EKF fail to capture the lane-keeping behavior of the vehicle because the GP-EKF is unable to account for the cross covariance information between the input and output of the GP driver model. The GP-EKF effectively assumes that the driver control actions are independent of the variance in the vehicle state for a given timestep. As a result, state-dependent control behaviors, such as lane keeping, are absent from the prediction.

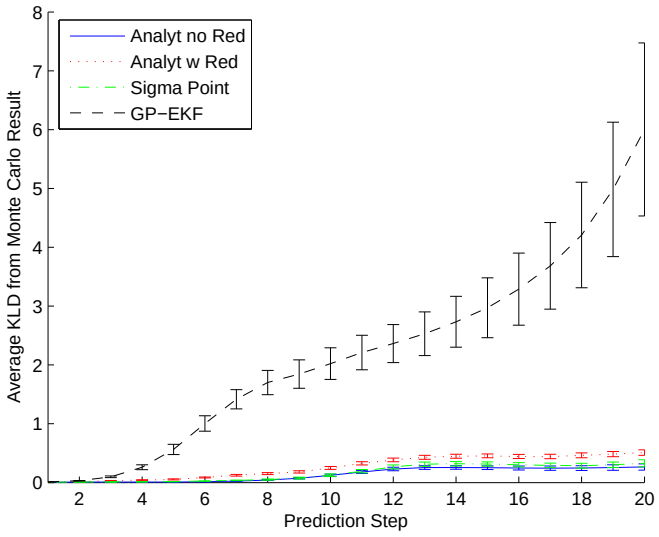
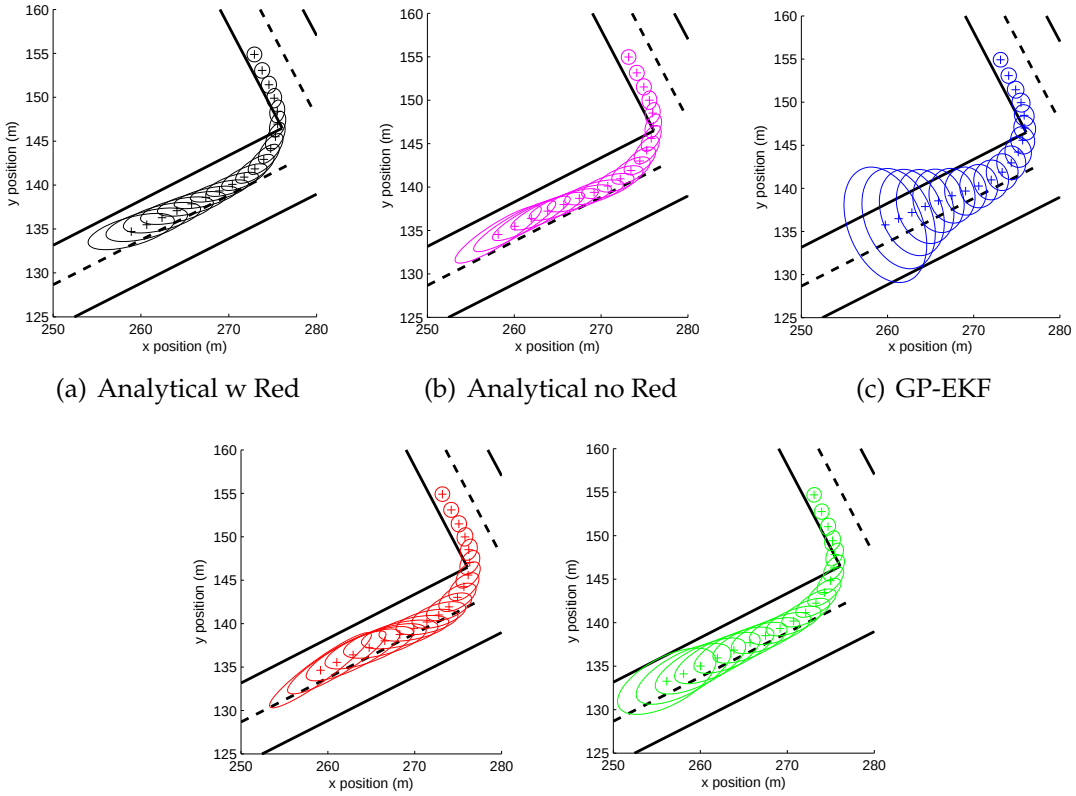


Figure 5.7: Average KLD from the Monte Carlo solution for the left turn maneuver.

Figure 5.7 shows the KLD results for the left turn maneuver. As expected, the analytical GP evaluation results with and without data selection and the sigma point prediction are all good approximations of the Monte Carlo result over the entire range of prediction timesteps. The GP-EKF approach performs much worse at later prediction timesteps when the lack of lane keeping causes large errors in the shape of the vehicle state distribution. This result matches the qualitative analysis of the typical trajectories in Figure 5.6.



(a) Analytical w Red (b) Analytical no Red (c) GP-EKF

(d) Sigma Point (e) Monte Carlo

Figure 5.8: Typical right turn prediction results for each algorithm over 20 timesteps. Ellipses represent the prediction error uncertainty at the 50% confidence interval.

Figure 5.8 shows typical prediction results and Figure 5.9 shows the average KLD results for the right turn maneuver. As with the left turn case, the GP-EKF without cross covariance terms is unable to capture lane keeping behavior, while the analytical and sigma point algorithms closely approximate the results from the Monte Carlo prediction.

Figure 5.10 shows the prediction results for the drive straight maneuver for each of the tested algorithms. In this case, all of the tested algorithms as well as the Monte Carlo prediction fail to capture noticeable lane keeping behavior.

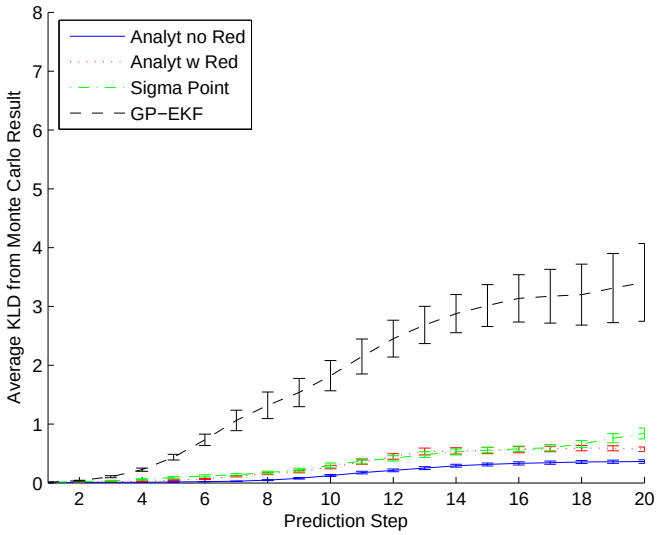
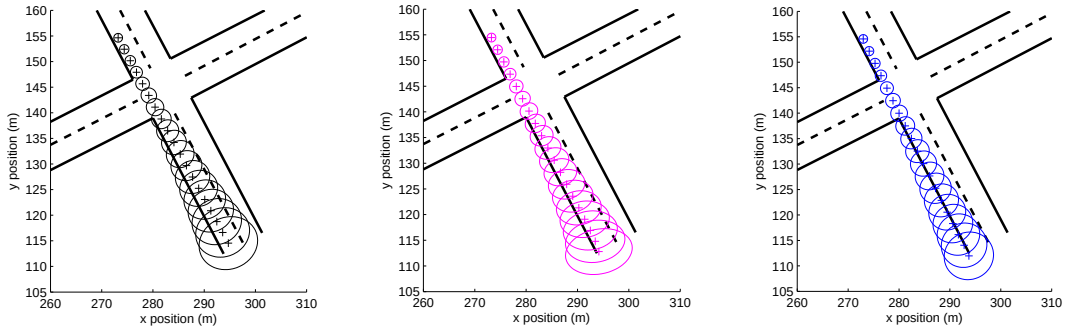


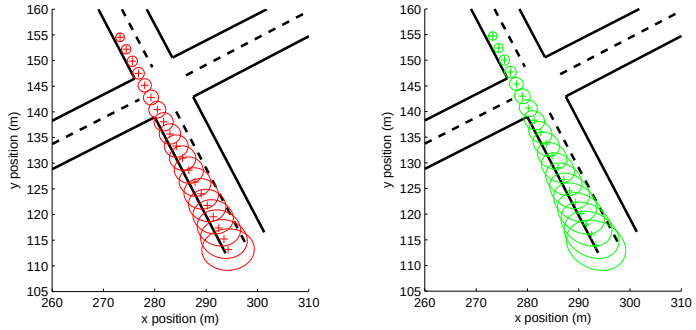
Figure 5.9: Average KLD from the Monte Carlo solution for the right turn maneuver.

This is likely due to a lack of lane-keeping behavior in the training data. During the left turn maneuver the driver must make constant steering corrections to stay in the correct driving lane. During the drive straight maneuver the drivers can align their vehicle with the driving lane and traverse the intersection with negligible additional steering input. This absence of steering information results in an GP steering model that effectively predicts a zero mean noisy steering output and prevents the prediction of any significant lane keeping behavior. The lack of cross covariance information in the GP-EKF has little effect in this case and the prediction results of all the algorithms closely match the Monte Carlo result.

Figure 5.11 presents the KLD results for the drive straight maneuver. The KLD for all four algorithms is very low across the entire prediction window, suggesting that all of the prediction algorithms closely match the Monte Carlo result for the drive straight maneuver. This agrees with the qualitative analysis



(a) Analytical w Red (b) Analytical no Red (c) GP-EKF



(d) Sigma Point (e) Monte Carlo

Figure 5.10: Drive straight prediction results for each algorithm over 20 timesteps. Ellipses represent the prediction error uncertainty at the 50% confidence interval.

of Figure 5.10.

Table 5.1 provides computation time statistics for each algorithm. Predictions were run in Matlab on a Core 2 Duo mobile processor running Windows 7. These results show that the analytical GP evaluation with on-the-fly data selection is nearly an order of magnitude faster than the sigma point algorithm and nearly two orders of magnitude faster than the analytical GP evaluation without data selection for all three maneuvers. The analytical GP evaluation algorithm with data selection is also faster than the GP-EKF due to the extra computation required to find the Jacobian matrix at each prediction step.

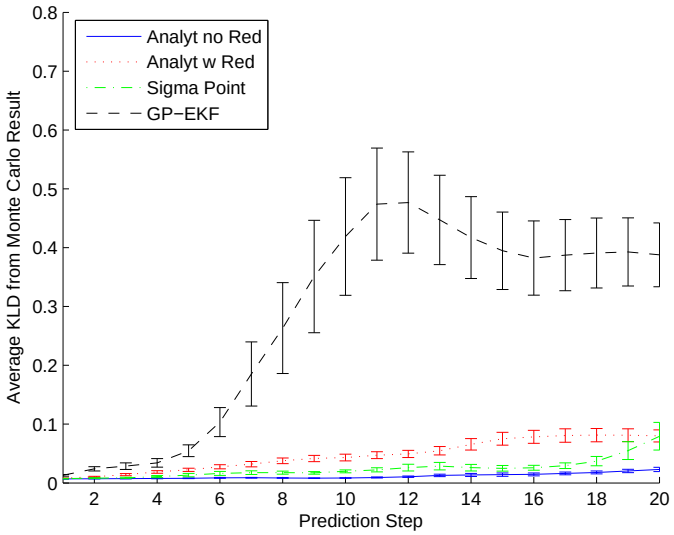


Figure 5.11: Average KLD from the Monte Carlo solution for the drive straight maneuver. Note: the axes for this figure have been rescaled for clarity.

5.8.2 Computational Scaling with Data Selection

The on-the-fly data selection method proposed in Section 5.4.3 was also evaluated using the left turn maneuver. Figure 5.12 presents the mean KLD of the analytical solution presented in Algorithm 1 as a function of the GP compression size n. The trend is as expected: for very small values of n, performance is poor, but for $n > 30$ there appears to be no correlation between propagation accuracy and the aggressiveness of the data selection.

Figure 5.13 plots the mean computation time for each prediction step as a function of n. For the range of data used in this paper, the prediction computation time scales with n^2, which reflects the computational complexity of the variance calculation. For very high values of n this scaling will be strongly influenced by the $\mathcal{O}(n^3)$ matrix inverse required to compute $\mathbf{K}_{\text{red}}^{-1}$. Figures 5.12

Table 5.1: Computation time statistics

	Left Turn		Straight		Right Turn	
Algorithm	μ_{comp}	σ_{comp}	μ_{comp}	σ_{comp}	μ_{comp}	σ_{comp}
Analyt w Red	0.109	0.003	0.111	0.015	0.114	0.013
Analyt no Red	7.382	0.036	7.375	0.054	7.422	0.056
GP-EKF	0.420	0.034	0.464	0.027	0.460	0.074
Sigma Point	1.051	0.015	1.072	0.020	1.074	0.016
units	s	s	s	s	s	s

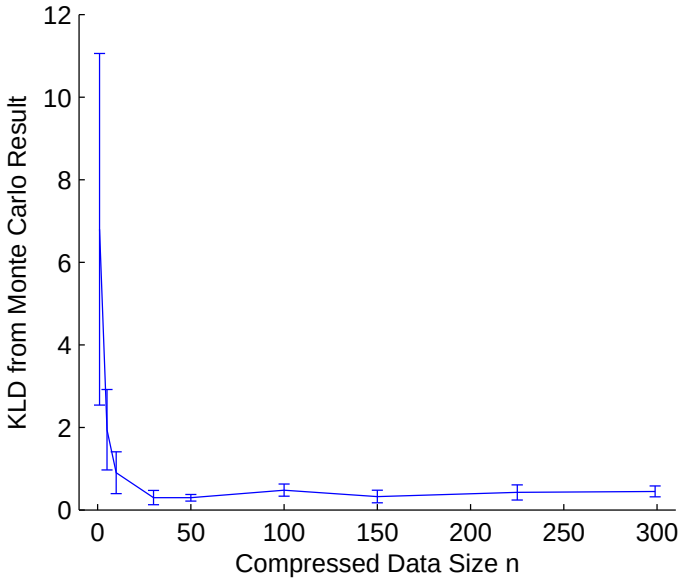


Figure 5.12: KLD from the Monte Carlo solution as a function of the on-the-fly data selection size n for the left turn maneuver.

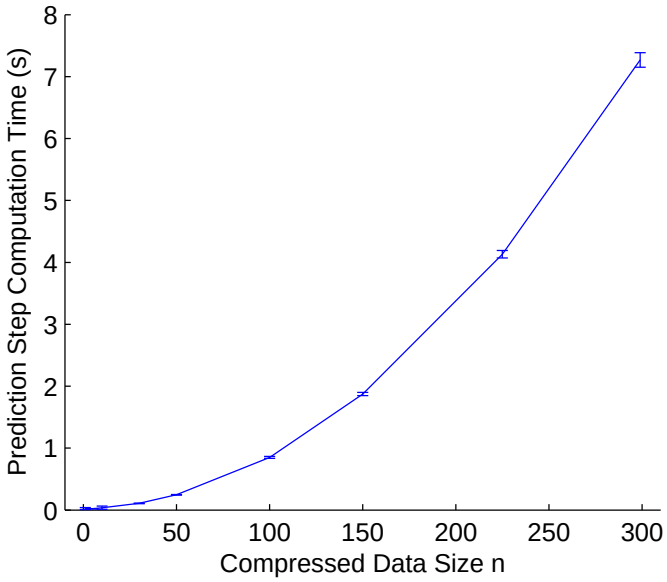


Figure 5.13: Computation time per prediction step as a function of the on-the-fly data selection size n for the left turn maneuver.

and 5.13 show that there is a large range of values for n such that computation time is greatly reduced without sacrificing propagation accuracy. For the value of $n = 30$ used above in this section, the impact of the data selection on propagation accuracy is negligible while the computation time is reduced by nearly two orders of magnitude.

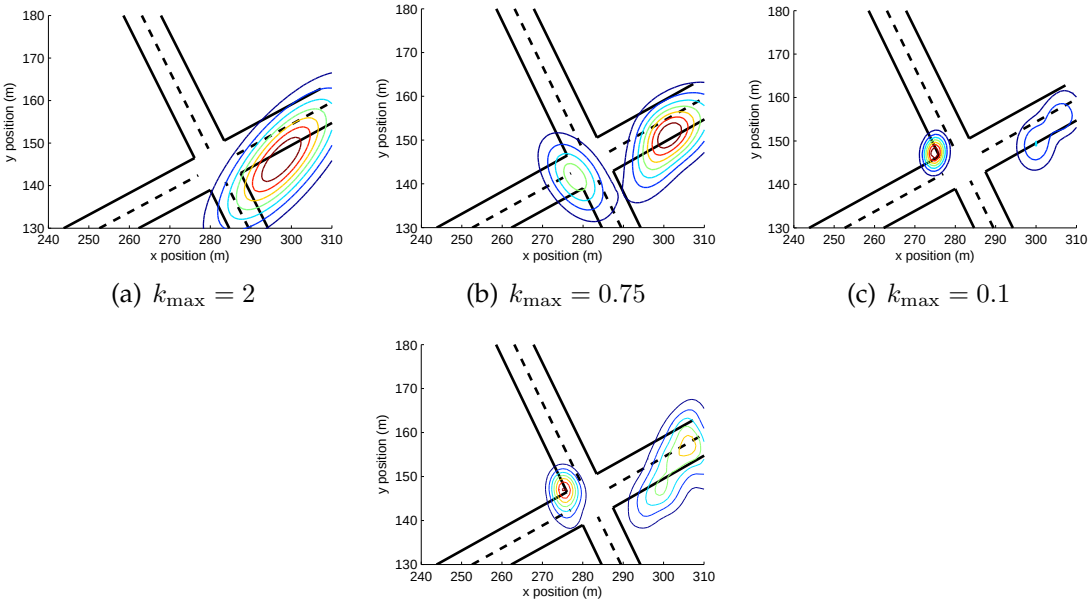
5.8.3 Prediction of Multimodal Behavior Using Adaptive Splitting

Identifying clear driving maneuvers in human driving data is not always possible. One common situation in which complex multimodal driving behavior occurs is when a driver is occasionally forced to stop and wait before execut-

ing a turn at an intersection. This behavior is a compound behavior that includes both a distinct stopping mode and a distinct turning mode. Prediction was performed using the adaptive GMM approach outlined in Section 5.6 for this type of combined driving maneuver. GP models for acceleration and turn rate were trained using intermixed training data from both stop-at-line and turn right or turn left maneuvers. Open loop prediction was then performed over 20 timesteps using the leave-one-out cross validation approach presented in Section 5.8.1. This prediction was performed for a range of maximum kurtosis splitting threshold values, $k_{\max} \in \{2, 1, 0.75, 0.5, 0.3, 0.1\}$, with a maximum recursion depth of $n_{\max}^{\text{rec}} = 2$. Gaussian mixture reduction is performed at the end of each prediction step using the mixture reduction approach proposed in [72]. This mixture reduction phase reduces the mixture down to a maximum of 15 elements at the end of each prediction step. The mixture is then reduced further as long as the Integral Squared Difference [85] of the reduced distribution stays within a predefined threshold of the original unreduced distribution. The ISD threshold used here is $\text{ISD}_{\max} = 1 \times 10^{-5}$.

Figure 5.14 shows the final predicted driver position distributions for the $k_{\max} = 2$, $k_{\max} = 0.75$, and $k_{\max} = 0.1$ cases, as well as the Monte Carlo prediction result using 1000 samples for a typical combined stop-at-line and turn left maneuver prediction trial. The $k_{\max} = 2$ case experiences no adaptive splitting and is unable to capture the bimodal nature of the system. The $k_{\max} = 0.75$ case captures the bimodal behavior but is unable to closely approximate the shape of the true distribution. The $k_{\max} = 0.1$ case captures the bimodal system behavior and provides a close approximation of the shape of the final distribution.

Figures 5.15 and 5.16 show the average KLD from the Monte Carlo result



(a) $k_{\max} = 2$ (b) $k_{\max} = 0.75$ (c) $k_{\max} = 0.1$

(d) Monte Carlo

Figure 5.14: Predicted distribution at step 20 using Monte Carlo sampling, $k_{\max} = 2$, $k_{\max} = 0.75$ and $k_{\max} = 0.1$.

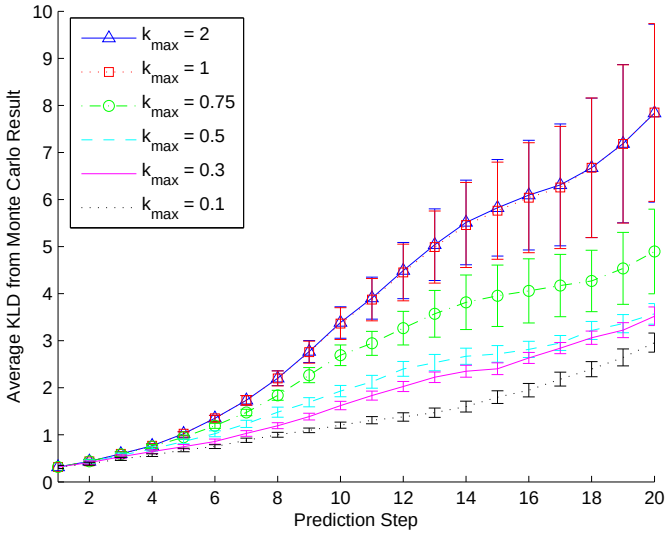


Figure 5.15: Average KLD from the Monte Carlo result for combined stop-at-line and turn left maneuvers.

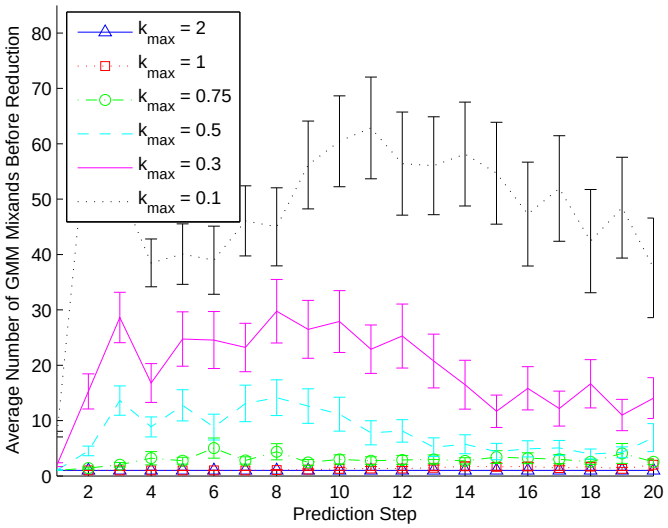


Figure 5.16: Average number of mixture elements before reduction for combined stop-at-line and turn left maneuvers.

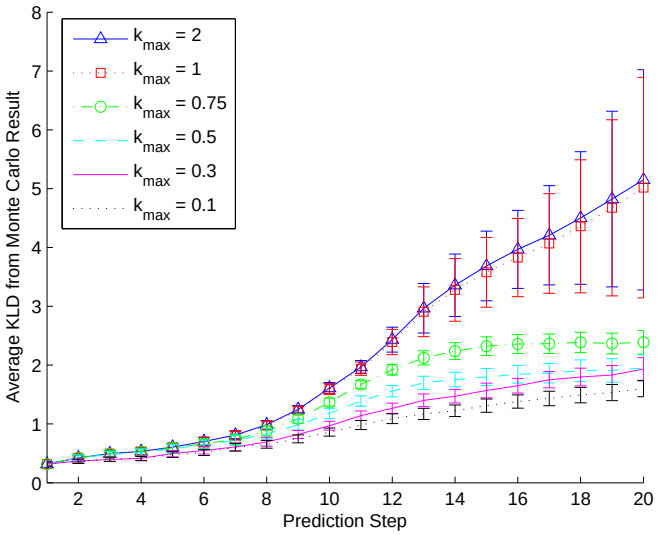


Figure 5.17: Average KLD from the Monte Carlo result for combined stop-at-line and turn right maneuvers.

and the average number of mixture elements before reduction at each prediction step for the combined stop-at-line and turn left maneuver over the range

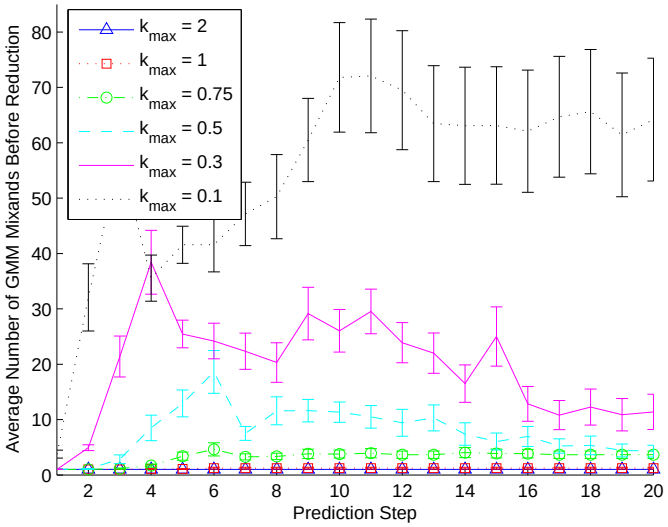


Figure 5.18: Average number of mixture elements before reduction for combined stop-at-line and turn right maneuvers.

of tested k_{max} values. Figures 5.17 and 5.18 show the average KLD from the Monte Carlo result and the average number of mixture elements before reduction for the combined stop-at-line and turn right maneuver. In each figure, error bars represent one Standard Error of the mean, $SE = \frac{\sigma}{\sqrt{n_{trial}}}$. The KLD between the Monte Carlo sample set and an analytically predicted GMM is computed using a Monte Carlo estimate, $D_{KL}(P||Q) \approx \sum_i \log(P(x_i)/Q(x_i))$, where $P(x_i)$ is approximated using Gaussian kernel density estimation. The $k_{max} = 2$ and $k_{max} = 1$ cases for both maneuvers experience little or no adaptive splitting and provide the poorest prediction performance. Prediction performance increases as the maximum kurtosis threshold is lowered, with a minimum KLD provided by the $k_{max} = 0.1$ case. This improvement in prediction performance comes at the cost of increased computational complexity, with the $k_{max} = 0.1$ case requiring a significantly higher number of mixture elements than the $k_{max} = 0.3$ case for a comparably small improvement in prediction

performance. The $k_{max} = 0.75$ and $k_{max} = 0.5$ splitting thresholds provide a good balance between prediction performance and computational complexity, especially for the stop-at-line and turn right maneuver.

Figure 5.19 presents the kurtosis computation time as a function of the on-the-fly data selection size n. The $\mathcal{O}(n^4)$ scaling of the kurtosis calculation makes its application with large data sizes prohibitive. On-the-fly data selection allows the excess kurtosis to be efficiently estimated using only a small set of the most relevant data points. The adaptive GMM trials in this section used a data selection size of $n = 10$ to compute the excess kurtosis splitting metric.

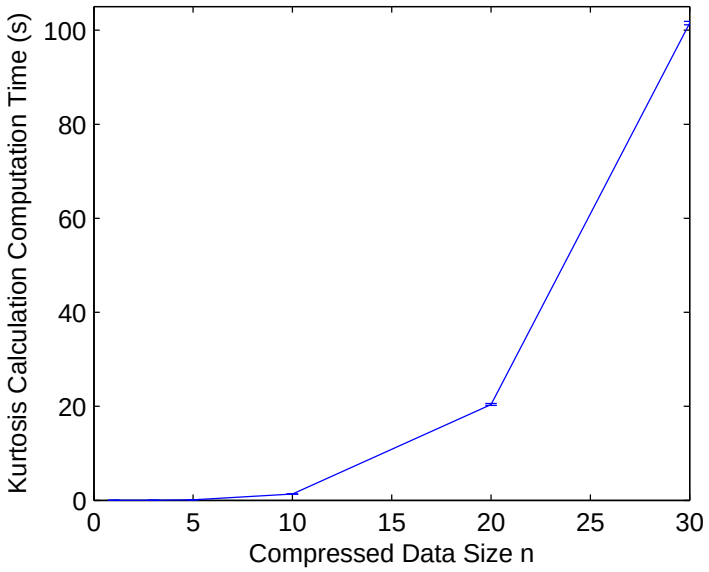


Figure 5.19: Kurtosis computation time as a function of the on-the-fly data selection size n.

5.9 Conclusion

This paper presents an algorithm for propagating a system state distribution through a two stage model consisting of both a GP regression model and a parametric system dynamics model. The presented algorithm relies on existing work to analytically evaluate GPs over an input distribution [11]. A novel on-the-fly data selection method is presented that greatly reduces the required computation time. To handle highly nonlinear and multimodal systems, an adaptive Gaussian mixture model formulation is presented that uses the excess kurtosis of the GP output as a measure of non-Gaussianity.

The two stage GP propagation algorithm is demonstrated on the problem of a human driver-vehicle model. The algorithm is shown to provide propagation accuracy similar to the pure sigma point approach, but with significant computational savings. For multimodal maneuvers, such as the case where the observed driver behavior includes both stopping and turning, the adaptive approach allows for predictions that accurately capture both driving behavior modes.

CHAPTER 6

CONCLUSIONS

A contingency path planner is developed to address limitations experienced by the reactive path planner used by Cornell's autonomous vehicle Skynet during the 2007 DARPA Urban Challenge. A discrete set of obstacle motion predictions is generated for each obstacle in the robot's environment based on the possible routes the obstacle might take along a predefined road network. A set of contingency plans is then optimized over all possible permutations of obstacle trajectories in the environment. Real-time planning performance is achieved through the use of an efficient spline-based path parameterization and a fast but accurate collision probability bound. A trajectory clustering technique is used to improve computational scaling and allow the contingency planning approach to scale to complex multi-obstacle environments.

This contingency planning approach is compared to the DUC planner and to an equivalent non-contingency planning approach using a series of human-in-the-loop experiments and simulations. A virtual obstacle interface is developed to enable the safe testing of dangerous collision avoidance scenarios while providing realistic two-way interactions with human driven obstacle vehicles. Testing results show that the contingency planning approach is much safer than the reactionary DUC planner and that it allows for increased aggressiveness compared to non-contingency planning, with no associated decrease in safety or reliability. Results from simulations using multiple human driven obstacles suggest that the contingency planner is safer than both the reactionary DUC planner and the human obstacle drivers.

An analytical prediction method is also developed for performing multi-step prediction of a two-stage system model. This model consists of a GP regression model for complex or poorly understood system dynamics and a parametric model for well understood dynamics. An on-the-fly data selection technique is developed to enable efficient computation of higher order moments of the GP output. An adaptive Gaussian mixture formulation is also developed to improve prediction accuracy for systems with highly nonlinear and multimodal behaviors. These prediction techniques are applied to a two-stage driver-vehicle model trained using driving data from a driving simulation environment. Results show significant computational advantages for analytical propagation using the on-the-fly data selection method and the adaptive Gaussian mixture model formulation allows for accurate prediction of multimodal compound behaviors.

APPENDIX A

KURTOSIS DERIVATION

The excess kurtosis of a distribution is defined as $\mathrm{kurt}(y) = \frac{E[y^4]}{(E[y^2])^2} - 3$. Using this definition, finding the excess kurtosis of the output of a GP is equivalent to finding the fourth order cumulant of the output, $\kappa_4 = \mu_4 - 3\mu_2^2$, and dividing by the squared variance, $\mathrm{kurt}(y) = \frac{\kappa_4}{(E[y^2])^2}$. The fourth order cumulant of a GP output evaluated over a Gaussian input distribution, $x^* \sim \mathcal{N}(\mu_x, \Sigma_x)$, is:

$$\begin{aligned}\kappa_4(g(x^*)) = & \\ & \kappa(\kappa_4(g(x^*)|x^*)) + 4\kappa(\kappa_3(g(x^*)|x^*), \kappa(g(x^*)|x^*)) \\ & + 3\kappa(\kappa_2(g(x^*)|x^*), \kappa_2(g(x^*)|x^*)) \\ & + 6\kappa(\kappa_2(g(x^*)|x^*), \kappa(g(x^*)|x^*), \kappa(g(x^*)|x^*)) \\ & + \kappa_4(\kappa(g(x^*)|x^*))\end{aligned} \tag{A.1}$$

where $\kappa_4(g(x^*)|x^*) = 0, \kappa_3(g(x^*)|x^*) = 0$ since the output of a GP at a known input point is Gaussian. This allows Equation A.1 to be simplified to:

$$\begin{aligned}\kappa_4(g(x^*)) = & \\ & 3\kappa(\sigma^2(x^*), \sigma^2(x^*)) + 6\kappa(\sigma^2(x^*), \mu(x^*), \mu(x^*)) \\ & + \kappa_4(\mu(x^*))\end{aligned} \tag{A.2}$$

The terms in Equation A.2 can then be expressed in terms of expectation integrals of the GP functions evaluated over x^*:

$$\begin{aligned}
&\kappa(\sigma^2(x^*), \sigma^2(x^*)) && \text{(A.3)}\\
&\quad = \text{var}(\sigma^2(x^*)) \\
&\quad = E[\sigma^2(x^*)^2] - E[\sigma^2(x^*)]E[\sigma^2(x^*)] \\
&\kappa(\sigma^2(x^*), \mu(x^*), \mu(x^*)) && \text{(A.4)}\\
&\quad = E[\mu(x^*)^2\sigma^2(x^*)] - 2E[\mu(x^*)\sigma^2(x^*)]E[\mu(x^*)] \\
&\qquad - E[\sigma^2(x^*)]E[\mu(x^*)^2] + 2E[\sigma^2(x^*)]E[\mu(x^*)]^2 \\
&\kappa_4(\mu(x^*)) = \mu_4[\mu(x^*)] - 3\mu_2[\mu(x^*)]^2 && \text{(A.5)}
\end{aligned}$$

where

$$\begin{aligned}
\mu_4(\mu(x^*)) =& E[\mu(x^*)^4] - 4E[\mu(x^*)^3]E[\mu(x^*)] && \text{(A.6)}\\
&+ 6E[\mu(x^*)^2]E[\mu(x^*)]^2 - 3E[\mu(x^*)]^4 \\
\mu_2(\mu(x^*)) =& E[\mu(x^*)^2] - E[\mu(x^*)]^2 && \text{(A.7)}
\end{aligned}$$

This means that computing the fourth order cumulant of the GP output requires the evaluation of the following additional expectation integrals that are not required for the evaluation of the mean, variance, and cross-covariance:

$$\begin{aligned}
&E[\mu(x^*)\sigma^2(x^*)] \\
&E[\mu(x^*)^3] \\
&E[\sigma^2(x^*)^2] \\
&E[\mu(x^*)^2\sigma^2(x^*)] \\
&E[\mu(x^*)^4]
\end{aligned}$$

of which $E[\mu(x^*)\sigma^2(x^*)]$ and $E[\mu(x^*)^3]$ are $\mathcal{O}(N^3)$ and the rest are $\mathcal{O}(N^4)$. These expectation integrals can be simplified by substituting in the GP equations for

$\mu(x^*)$ and $\sigma^2(x^*)$:

$$E[\mu(x^*)\sigma^2(x^*)] = \sigma_n^2 \sum_i \beta_i l_i - \sum_i \sum_j \sum_k \mathbf{K}_{ij}^{-1} \beta_k O_{ijk} \tag{A.8}$$

$$E[\mu(x^*)^3] = \sum_i \sum_j \sum_k \beta_i \beta_j \beta_k O_{ijk} \tag{A.9}$$

$$E[\sigma^2(x^*)^2] = (\sigma_n^2)^2 - 2\sigma_n^2 \sum_i \sum_j \mathbf{K}_{ij}^{-1} L_{ij} + \tag{A.10}$$

$$\sum_i \sum_j \sum_k \sum_d \mathbf{K}_{ij}^{-1} \mathbf{K}_{kd}^{-1} W_{ijkd}$$

$$E[\mu(x^*)^2\sigma^2(x^*)] = \sigma_n^2 \sum_i \sum_j \beta_i \beta_j L_{ij} - \tag{A.11}$$

$$\sum_i \sum_j \sum_k \sum_d \mathbf{K}_{ij}^{-1} \beta_k \beta_d W_{ijkd}$$

$$E[\mu(x^*)^4] = \sum_i \sum_j \sum_k \sum_d \beta_i \beta_j \beta_k \beta_d W_{ijkd} \tag{A.12}$$

where $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_N]^T = \mathbf{K}^{-1}\mathbf{y}$ and O_{ijk} and W_{ijkd} are defined as:

$$O_{ijk} \tag{A.13}$$
$$= \int C(x^*, x_i)\, C(x^*, x_j)\, C(x^*, x_k)\, \mathcal{N}(x^*|\mu_x, \Sigma_x) dx^*$$

$$W_{ijkd} \tag{A.14}$$
$$= \int C(x^*, x_i)\, C(x^*, x_j)\, C(x^*, x_k)\, C(x^*, x_d)\, \mathcal{N}(x^*|\mu_x, \Sigma_x) dx^*$$

Since the GP Covariance function, $C(x_i, x_j)$, is of the squared exponential form, the integral expressions for O_{ijk} and W_{ijkd} can be expressed as the integral over the product of four and five Gaussian distributions respectfully, with the appropriate normalization terms. Since the product of Gaussian distributions is itself

a Gaussian, the above expressions simplify to:

$$O_{ijk} = z_O \int \mathcal{N}(x^*|\bar{\mu}_O, \bar{\Sigma}_O)dx^* \tag{A.15}$$

$$= z_O$$

$$W_{ijkd} = z_W \int \mathcal{N}(x^*|\bar{\mu}_W, \bar{\Sigma}_W)dx^* \tag{A.16}$$

$$= z_W$$

where $\bar{\Sigma}_O = (3\Lambda^{-1} + \Sigma_x^{-1})^{-1}$ and $\bar{\Sigma}_W = (4\Lambda^{-1} + \Sigma_x^{-1})^{-1}$. The normalization terms, z_O and z_W, are defined as:

$$\begin{aligned} z_O = (\sigma_f^2)^3 \left|\bar{\Sigma}_O\right|^{\frac{1}{2}} |\Sigma_x|^{-\frac{1}{2}} \exp(&-\frac{1}{2}(x_i - x_j)^T B_1^O (x_i - x_j) \\ &- \frac{1}{2}(x_i - x_k)^T B_1^O (x_i - x_k) - \frac{1}{2}(x_j - x_k)^T B_1^O (x_j - x_k) \\ &- \frac{1}{2}(x_i - \mu_x)^T B_2^O (x_i - \mu_x) \\ &- \frac{1}{2}(x_j - \mu_x)^T B_2^O (x_j - \mu_x) - \frac{1}{2}(x_k - \mu_x)^T B_2^O (x_k - \mu_x)) \end{aligned} \tag{A.17}$$

$$\begin{aligned} z_W = (\sigma_f^2)^4 \left|\bar{\Sigma}_W\right|^{\frac{1}{2}} |\Sigma_x|^{-\frac{1}{2}} \exp(&-\frac{1}{2}(x_i - x_j)^T B_1^W (x_i - x_j) \\ &- \frac{1}{2}(x_i - x_k)^T B_1^W (x_i - x_k) - \frac{1}{2}(x_j - x_k)^T B_1^W (x_j - x_k) \\ &- \frac{1}{2}(x_i - \mu_x)^T B_2^W (x_i - \mu_x) \\ &- \frac{1}{2}(x_j - \mu_x)^T B_2^W (x_j - \mu_x) - \frac{1}{2}(x_k - \mu_x)^T B_2^W (x_k - \mu_x)) \end{aligned} \tag{A.18}$$

where $B_1^O = \Lambda^{-1}\bar{\Sigma}_O\Lambda^{-1}$, $B_1^W = \Lambda^{-1}\bar{\Sigma}_W\Lambda^{-1}$, $B_2^O = \Lambda^{-1}\bar{\Sigma}_O\Sigma_x^{-1}$, $B_2^O = \Lambda^{-1}\bar{\Sigma}_O\Sigma_x^{-1}$.

BIBLIOGRAPHY

[1] O. Aichholzer, F. Aurenhammer, D. Alberts, and B. Gaertner. A Novel Type of Skeleton for Polygons. *Journal of Universal Computer Science*, 1(12):752–761, 1995.

[2] S. Alfano. Addressing Nonlinear Relative Motion For Spacecraft Collision Probability. *AIAA Paper*, (2006-6760):15, 2006.

[3] M. Althoff, O. Stursberg, and M. Buss. Model-based probabilistic collision detection in autonomous driving. *Intelligent Transportation Systems, IEEE Transactions on*, 10(2):299–310, 2009.

[4] G.S. Aoude, B.D. Luders, D.S. Levine, and J.P. How. Threat-aware Path Planning in Uncertain Urban Environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan*, 2010.

[5] J. Bellingham, A. Richards, and J.P. How. Receding horizon control of autonomous aerial vehicles. In *American Control Conference, 2002. Proceedings of the 2002*, volume 5, pages 3741–3746. IEEE, 2002. ISBN 0780372980.

[6] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun. Learning motion patterns of people for compliant robot motion. *The International Journal of Robotics Research*, 24(1):31–48, 2005.

[7] L. Blackmore, H. Li, and B. Williams. A probabilistic approach to optimal robust path planning with obstacles. In *American Control Conference, 2006*, pages 7–pp. IEEE, 2006. ISBN 1424402093.

[8] P.T. Boggs and J.W. Tolle. Sequential quadratic programming. *Acta numerica*, 4(-1):1–51, 1995.

[9] Alberto Broggi, Pietro Cerri, Mirko Felisa, Maria Chiara Laghi, Luca Mazzei, and Pier Paolo Porta. The vislab intercontinental autonomous challenge: an extensive test for a platoon of intelligent vehicles. *International Journal of Vehicle Autonomous Systems*, 10(3):147–164, 2012.

[10] M. Buehler, K. Iagnemma, and S. Singh. *The Darpa Urban Challenge: Autonomous Vehicles in City Traffic*. Springer Verlag, 2010.

[11] J.Q. Candela, A. Girard, J. Larsen, and C.E. Rasmussen. Propagation of uncertainty in bayesian kernel models-application to multiple-step ahead forecasting. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, volume 2, pages II–701. IEEE, 2003.

[12] R. Coulter. Implementation of the pure pursuit path tracking algorithm. Technical Report CMU-RI-TR-92-01, The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, January 1992.

[13] L. Cremean, T. Foote, J. Gillula, G. Hines, D. Kogan, K. Kriechbaum, J. Lamb, J. Leibs, L. Lindzey, C. Rasmussen, A. Stewart, J. Burdick, and R. Murray. Alice: An information-rich autonomous vehicle for high-speed desert navigation. *Journal of Field Robotics*, 23(9):777–810, 2006.

[14] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. Path planning for autonomous driving in unknown environments. In *Experimental Robotics*, pages 55–64. Springer, 2009.

[15] E. Espié, C. Guionneau, B. Wymann, C. Dimitrakakis, R. Coulom, and A. Sumner. Torcs-the open racing car simulator, 2005. URL `http://torcs.sourceforge.net/`.

[16] Eric Espié, Christophe Guionneau, Bernhard Wymann, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. TORCS, the open racing car simulator. http://www.torcs.org, 2006. URL `http://www.torcs.org`.

[17] D. Ferguson, M. Darms, C. Urmson, and S. Kolski. Detection, prediction, and avoidance of dynamic obstacles in urban environments. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, 2008.

[18] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998. ISSN 0278-3649.

[19] Martin Fischer, Håkan Sehammar, Mikael Ljung Aust, Martin Nilsson, and H Weiefors. Advanced driving simulators as a tool in early development phases of new active safety functions. *Advances in Transportation Studies*, (Special issue), 2011.

[20] Luke Fletcher, Seth Teller, Edwin Olson, David Moore, Yoshiaki Kuwata, Jonathan How, John Leonard, Isaac Miller, Mark Campbell, Dan Huttenlocher, Aaron Nathan, and Frank-Robert Kline. The MIT-Cornell collision and why it happened. *Journal of Field Robotics*, 25(10):775–807, 2008.

[21] A. Forsgren, P.E. Gill, and M.H. Wright. Interior methods for nonlinear optimization. *SIAM Review*, 44(4):525–597, 2002.

[22] Z. Fu, W. Hu, and T. Tan. Similarity based vehicle trajectory clustering and anomaly detection. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, volume 2. IEEE, 2005. ISBN 0780391349.

[23] S. Gaffney and P. Smyth. Trajectory clustering with mixtures of regression models. In *Proceedings of the fifth ACM SIGKDD international confer-*

ence on Knowledge discovery and data mining, pages 63–72. ACM, 1999. ISBN 1581131437.

[24] Giancarlo Genta. *Motor Vehicle Dynamics: Modeling and Simulation*. World Scientific, Singapore, 1997. ISBN 9810229119.

[25] R. Geraerts and M.H. Overmars. Creating high-quality paths for motion planning. *The International Journal of Robotics Research*, 26(8):845–863, 2007.

[26] Thomas D. Gillespie. *Fundamentals of Vehicle Dynamics*. Society of Automotive Engineers, Inc., Warrendale, Pennsylvania, 1992.

[27] R.L. Graham. An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. *Information Processing Letters*, 1(4):132–133, 1972.

[28] P. Groot, P. Lucas, and P. van den Bosch. Multiple-step time series forecasting with sparse gaussian processes. In *Causmaecker, P. De (ed.), Proceedings of the 23rd Benelux Conference on Artificial Intelligence*, page 1. Sl: sn, 2011.

[29] Erico Guizzo. How googles self-driving car works. *IEEE Spectrum Online, October*, 18, 2011.

[30] J. Hardy and M. Campbell. Contingency planning over probabilistic hybrid obstacle predictions for autonomous road vehicles. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2237–2242. IEEE, 2010.

[31] J. Hardy and M. Campbell. Clustering obstacle predictions to improve contingency planning for autonomous road vehicles in congested environments. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1605–1611. IEEE, 2011.

[32] Jason Hardy and Mark Campbell. Contingency planning over probabilistic obstacle predictions for autonomous road vehicles. *IEEE Transactions on Robotics*, 2013.

[33] Jason Hardy, Mark Campbell, Isaac Miller, and Brian Schimpf. Sensitivity analysis of an optimization-based trajectory planner for autonomous vehicles in urban environments. *Unmanned/Unattended Sensors and Sensor Networks V*, 7112(1):711211, 2008. doi: 10.1117/12.802599. URL `http://link.aip.org/link/?PSI/7112/711211/1`.

[34] Jason Hardy, Frank Havlak, and Mark Campbell. Multiple-step prediction using adaptive gaussian mixtures for a two stage gaussian process model. *Journal of Field Robotics*, In Preperation.

[35] Jason Hardy, Frank Havlak, Jon Schoenberg, Theo Park, and Mark Campbell. Experimental evaluation of a contingency based path planner for autonomous driving. *Journal of Field Robotics*, In Preperation.

[36] Frank Havlak and Mark Campbell. Discrete and continuous, probabilistic anticipation for autonomous robots in urban environments. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 7833, page 12, 2010.

[37] I. Hwang and CE Seah. Intent-Based Probabilistic Conflict Detection for the Next Generation Air Transportation System. *Proceedings of the IEEE*, 96 (12):2040–2059, 2008.

[38] Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4):411–430, 2000.

[39] Lamia Iftekhar and Reza Olfati-Saber. Autonomous driving for vehicular networks with nonlinear dynamics. In *Intelligent Vehicles Symposium (IV), 2012 IEEE*, pages 723–729. IEEE, 2012.

[40] Joshua Joseph, Finale Doshi-Velez, AlbertS. Huang, and Nicholas Roy. A bayesian nonparametric approach to modeling motion patterns. *Autonomous Robots*, 31(4):383–400, 2011. ISSN 0929-5593. doi: 10.1007/s10514-011-9248-x. URL http://dx.doi.org/10.1007/s10514-011-9248-x.

[41] S.J. Julier and J.K. Uhlmann. A general method for approximating nonlinear transformations of probability distributions. *Robotics Research Group, Department of Engineering Science, University of Oxford, Oxford, OC1 3PJ United Kingdom, Tech. Rep*, 1996.

[42] S.J. Julier and J.K. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls*, volume 3, page 26. Citeseer, 1997.

[43] Shin Kato, Sadayuki Tsugawa, Kiyohito Tokuda, Takeshi Matsui, and Haruki Fujii. Vehicle control algorithms for cooperative driving with automated vehicles and intervehicle communications. *Intelligent Transportation Systems, IEEE Transactions on*, 3(3):155–161, 2002.

[44] K. Kim, D. Lee, and I. Essa. Gaussian process regression flow for analysis of motion trajectories. In *Proceedings of the 2011 IEEE International Conference on Computer Vision (ICCV)*, pages 1164–1171, 2011.

[45] J. Ko and D. Fox. Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90, 2009.

[46] Daniel E. Koditschek and Elon Rimon. Robot Navigation Functions on Manifolds with Boundary. *Advances in Applied Mathematics*, 11:412–442, 1990.

[47] A. Kushleyev and M. Likhachev. Time-bounded lattice for efficient planning in dynamic environments. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, pages 4303–4309. IEEE Press, 2009.

[48] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J.P. How. Real-time Motion Planning with Applications to Autonomous Urban Driving. *IEEE Transactions on control systems technology*, 17(5):1105, 2009.

[49] A. Lambert, D. Gruyer, GS Pierre, and A.N. Ndjeng. Collision probability assessment for speed control. In *Intelligent Transportation Systems, 2008. ITSC 2008. 11th International IEEE Conference on*, pages 1043–1048. IEEE, 2008.

[50] J.C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[51] C. Laugier, I.E. Paromtchik, C. Tay, K. Mekhnacha, G. Othmezouri, and H. Yanagihara. Collision risk assessment to improve driving safety.

[52] S. LaValle. Rapidly-exploring random trees: A new tool for path planning, 1998.

[53] S. LaValle and J. Ku. Randomized kinodynamic planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, 1999.

[54] J.G. Lee, J. Han, and K.Y. Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 593–604. ACM, 2007.

[55] M.B. Milam, K. Mushambi, and R.M. Murray. A new computational approach to real-time trajectory generation for constrained mechanical systems. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, volume 1, pages 845–851. IEEE, 2000. ISBN 0780366387.

[56] I. Miller and M. Campbell. Rao-blackwellized particle filtering for mapping dynamic environments. In *Proceedings of the 2007 International Conference on Robotics and Automation*, pages 3862 – 3869, April 2007.

[57] I. Miller and M. Campbell. Particle filtering for map-aided localization in sparse gps environments. In *Proceedings of the 2008 International Conference on Robotics and Automation*, pages 1834 – 1841, May 2008.

[58] I. Miller, B. Schimpf, J. Leyssens, and M. Campbell. Tightly-coupled gps / ins system design for autonomous urban navigation. In *Proceedings of the 2008 IEEE / ION Position, Localization, and Navigation Symposium*, May 2008.

[59] Isaac Miller, Sergei Lupashin, Noah Zych, Pete Moran, Brian Schimpf, Aaron Nathan, and Ephrahim Garcia. Cornell University's 2005 DARPA Grand Challenge Entry. *Journal of Field Robotics*, 23(8):625–652, 2006.

[60] Isaac Miller, Mark Campbell, Dan Huttenlocher, Frank-Robert Kline, Aaron Nathan, Jason Catlin Sergei Lupashin, Brian Schimpf, Pete Moran, Noah Zych, Ephrahim Garcia, Mike Kurdziel, and Hikaru Fujishima. Team cornell's skynet: Robust perception and planning in an urban environment. *Journal of Field Robotics*, 25(8):493–527, 2008.

[61] Isaac Miller, Mark Campbell, and Daniel Huttenlocher. Efficient, unbiased tracking of multiple dynamic obstacles under large viewpoint changes. *IEEE Transactions on Robotics*, 27(1):29–46, 2011.

[62] Isaac Miller, Mark E. Campbell, and Dan Huttenlocher. Map-aided localization in sparse global positioning system environments using vision and particle filtering. *J. Field Robotics*, 28(5):619–643, 2011.

[63] J. Miura and Y. Shirai. Probabilistic uncertainty modeling of obstacle motion for robot motion planning. *Journal of Robotics and Mechatronics*, 14(4): 349–356, 2002.

[64] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, and D. Haehnel. Junior: The stanford entry in the urban challenge. *Journal of Field Robotics*, 25(9):569–597, 2008.

[65] R.A. Paielli, H. Erzberger, and Ames Research Center. Conflict probability estimation for free flight. *Journal of Guidance, Control, and Dynamics*, 20(3): 588–596, 1997.

[66] Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[67] G. Prokop. Modeling human vehicle driving by model predictive online optimization. *Vehicle System Dynamics*, 35(1):19–53, 2001.

[68] Mark L Psiaki, Jonathan R Schoenberg, and Isaac T Miller. Gaussian mixture approximation by another gaussian mixture for "blob" filter resampling. In *2010 AIAA Guidance, Navigation, and Control Conference*, 2010.

[69] J. Quinonero-Candela, A. Girard, and C.E. Rasmussen. *Prediction at an Uncertain Input for Gaussian Processes and Relevance Vector Machines Application to Multiple-Step Ahead Time-Series Forecasting*. IMM, Informatik og Matematisk Modelling, DTU, 2003.

[70] C.E. Rasmussen and C.K.I. Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, MA, 2006.

[71] Tom Robinson, Eric Chan, and Erik Coelingh. Operating platoons on public motorways: An introduction to the sartre platooning programme. In *17th World Congress on Intelligent Transport Systems*, 2010.

[72] Andrew R Runnalls. Kullback-leibler approach to gaussian mixture reduction. *Aerospace and Electronic Systems, IEEE Transactions on*, 43(3):989–999, 2007.

[73] R. Schubert, C. Adam, M. Obst, N. Mattern, V. Leonhardt, and G. Wanielik. Empirical evaluation of vehicular models for ego motion estimation. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 534–539. IEEE, 2011.

[74] Wing Ip Tam, KN Plataniotis, and D Hatzinakos. An adaptive gaussian sum algorithm for radar tracking. *Signal processing*, 77(1):85–104, 1999.

[75] M. Tay and C. Laugier. Modelling smooth paths using gaussian processes. In *Field and Service Robotics*, pages 381–390. Springer, 2008.

[76] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, et al. Stanley: The robot that won the DARPA Grand Challenge: Research Articles. *Journal of Robotic Systems*, 23(9):692, 2006.

[77] Sebastian Thrun and Arno Bucken. Integrating grid-based and topological maps for mobile robot navigation. In *AAAI/IAAI, Vol. 2*, pages 944–950, 1996.

[78] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert,

T. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, B. Salesky, Y.-W. Seo, S. Singh, J. Snider, A. Stentz, W. Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolau, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, and D. Ferguson. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):426–466, 2008.

[79] Robert J. Vanderbei, David, and F. Shanno. An interior-point algorithm for nonconvex nonlinear programming. *Computational Optimization and Applications*, 13:231–252, 1999.

[80] A. Wächter and L.T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

[81] A. Wächter and L.T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. ISSN 0025-5610.

[82] J. Wang, D. Fleet, and A. Hertzmann. Gaussian process dynamical models. *Advances in neural information processing systems*, 18:1441, 2006.

[83] Stephen Waydo and Richard M. Murray. Vehicle Motion Planning Using Stream Functions. In *IEEE International Conference on Robotics and Automation*, 2003.

[84] R. Wein, J. Van Den Berg, and D. Halperin. Planning high-quality paths and corridors amidst obstacles. *The International Journal of Robotics Research*, 27 (11-12):1213–1231, 2008.

[85] J.L. Williams and P.S. Maybeck. Cost-function-based gaussian mixture reduction for target tracking. In *Information Fusion, 2003. Proceedings of the Sixth International Conference of*, volume 2, pages 1047–1054, 2003. doi: 10.1109/ICIF.2003.177354.

[86] J. Wit. *Vector Pursuit Path Tracking for Autonomous Ground Vehicles*. PhD thesis, University of Florida, 2000.

[87] J. Y. Wong. *Theory of Ground Vehicles*. John Wiley & Sons, Inc., New York, New York, 3rd edition, 2001.

[88] Brian D Ziebart, Nathan Ratliff, Garratt Gallagher, Christoph Mertz, Kevin Peterson, J Andrew Bagnell, Martial Hebert, Anind K Dey, and Siddhartha Srinivasa. Planning-based prediction for pedestrians. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 3931–3936. IEEE, 2009.